UNIVERSIDAD DE ALCALÁ

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

Trabajo de Fin de Grado

Implementación de un módulo para el análisis y diseño de guías de onda en un software de simulaciones electromagnéticas

Autor: Daniel Rodríguez Martínez

Directora: Lorena Lozano Plata

Tribunal:

Presidente: .....

Vocal 1: .....

Vocal 2: .....

Fecha: .....

A mi familia, y muy especialmente a mi madre, por confiar tanto en mí durante todo este tiempo. A los amigos con los que tanto tiempo he pasado estos años; a Alejandro, Román, Adri y José. Y también al grupo de electromagnetismo computacional por permitirme trabajar con su software, adquirir experiencia y darme ganas de conocer más.

Un buen algoritmo es como un cuchillo afilado: hace exactamente lo que debe hacer con el mínimo esfuerzo aplicado. Usar el algoritmo incorrecto para resolver un problema es como cortar un filete con un destornillador: seguramente obtengas algo comestible, pero necesitarás más esfuerzo que lo normal y el resultado no será tan agradable

Thomas H. Cormen et al.

Índice general

Índice de figuras

Índice de tablas

Resumen

Resumen

El presente trabajo de fin de grado trata el desarrollo de un módulo para efectuar cálculos de guía de onda en la herramienta de simulación electromagnética newFASANT. Se realizará un diseño de los componentes que tendrá que tener la aplicación y se ofrecerá una implementación que haga uso de los conceptos teóricos sobre guías de onda y alimentación por puertos a fin de poder realizar estudios posteriores sobre las guías.

Abstract

This final degree project will present the development of a waveguide calculation module that will be added to the EM simulation software newFASANT. A design of the components that will be added will be presented, and a final implementation will be developed applying theorical concepts about waveguides in order to create a tool that allows to analyze waveguide parameters.

Palabras clave

Simulación, Electromagnética, Guías de onda

Resumen extendido

Durante los últimos 50 años el desarrollo de la sociedad ha avanzado con la mejora de las comunicaciones humanas. Actualmente medios como el teléfono, la televisión o la radio dependen de las cualidades de las ondas electromagnéticas como medio de propagación para transmitir a largas distancias de forma inalámbrica información que permita mantenernos comunicados.

En este sentido, las guías de onda son estructuras ampliamente extendidas debido a las posibilidades que ofrecen para ayudar a la propagación de ondas electromagnéticas de alta frecuencia, que son aquellas en las que estamos interesados debido a las prestaciones que ofrecen para poder transmitir información en ellas.

Desarrollar sistemas electromagnéticos de transmisión de datos requiere un largo proceso de estudio con el fin de hacerlos eficientes y de permitir transmitir correctamente la información. Este estudio requiere conocer cómo funcionan las estructuras que propagan las ondas y las propias ondas.

Antiguamente esto requería mucho tiempo manual de cálculo y bastante coste. A día de hoy, las capacidades computacionales han avanzado lo suficiente como para disponer de herramientas que ayuden a realizar estos cálculos y que sirvan de apoyo a los ingenieros que trabajan con estos sistemas.

Para ello se hará uso de un software de electromagnetismo computacional llamado newFASANT, que dispone de tecnologías que permiten realizar cálculos y simulaciones electromagnéticas, y se aprovecharán estas tecnologías para desarrollar un módulo integrado que permita trabajar con guías de onda.

1 Introducción

Durante los últimos 50 años la sociedad ha experimentado dos cambios profundos nunca antes imaginados y que han cambiado completamente nuestra forma de vida respecto a cualquier época anterior.

En primer lugar, la innovación en el mundo de las comunicaciones, permitiendo a la sociedad disponer de múltiples formas de estar en contacto los unos con los otros y facilitando de este modo la convivencia y el acceso a la información. Las comunicaciones a día de hoy nos permiten acceder a todo un mundo de información y estar en contacto con nuestros seres más queridos.

Y en segundo lugar, en parte, gracias a esa innovación, el desarrollo de la computación y de la informática, permitiendo la construcción de máquinas electrónicas capaces de realizar millones de cálculos por segundo, con unas capacidades que no hacen más que mejorar año a año y que han permitido mejorar tecnológicamente a la sociedad, facilitando en gran medida el trabajo en determinados sectores.

Este gran avance de las comunicaciones es imposible que se produzca de forma instantánea, sin la inversión en una infraestructura que la soporte. A día de hoy, las técnicas, los medios y los sistemas que permiten la comunicación son muchos y muy dispares, desde la comunicación por enlaces de fibra óptica hasta la clásica comunicación por radio, que lejos de ser la misma que en sus primeros momentos ha evolucionado hasta conseguir dar soporte a una compleja rama de servicios.

Históricamente, la radio, el telégrafo, y posteriormente la televisión fueron los primeros sistemas de comunicación que aprovecharon las capacidades de las ondas electromagnéticas. Sin embargo, a día de hoy, otros medios de mayor alcance, como los sistemas de telefonía o de transmisión de paquetes de datos (incluyendo tecnologías como GPRS, UMTS, HSDPA, LTE...) también dependen de estas ondas electromagnéticas. Por supuesto, otros muchos sistemas más orientados al ámbito militar que al civil, como los sistemas de radar y de HF también utilizan este mismo medio.

Uno de los medios más utilizados para sostener todo ese sistema de radio son las guías de onda. Las guías de onda, sobre las que se basa este trabajo, son las estructuras que se usan para guiar las ondas electromagnéticas usadas por estos sistemas de radio. El uso de guías de onda es casi tan antiguo como los propios sistemas de radio.

No obstante, diseñar y construir guías de onda no es una tarea sencilla. En algunas ocasiones, resulta complejo adaptarse al entorno al que va a estar expuesta la guía de

onda. Por ejemplo, las guías de onda que pueden acoplarse a un portaaviones situado en el lugar mas recógnito del planeta o incluso las que puede utilizar un satélite de comunicaciones. Diseñar y probar antenas en este tipo de situaciones no siempre es sencillo, por lo que uno de los principales objetivos de la ciencia durante las últimas décadas es el de aprovechar el potencial que ofrece la computación para simular esos entornos de forma virtual por medio de programas informáticos que permitan diseñar, probar, medir y validar elementos electromagnéticos como las guías de onda antes de construirlas, de modo que una vez esa guía de onda se construya de forma física se haga con la confianza de que desempeñe su función correctamente.

En parte, otro motivo que impulsa esta necesidad de poder realizar simulaciones de determinados proyectos científicos de este tipo, es el coste que supone el diseño y la construcción de estos elementos electromagnéticos. Las empresas que investigan en estos sectores desean optimizar la inversión que realizan en el desarrollo de estos elementos electromagnéticos y no están interesadas en emplear recursos para antenas que se encuentren defectuosas o que no sean capaces de hacer correctamente su función. Es por ello que estas empresas están interesadas en soluciones que les permita simular el desempeño que estos sistemas electromagnéticos pueden ofrecer antes de construir dichas antenas con el fin de optimizar su diseño.

De este modo, los ingenieros encargados del diseño de estos sistemas ahorrarían tiempo al desarrollo de los proyectos, apoyándose en una solución de software para realizar cálculos, mediciones y pruebas que de otro modo requeriría de cara al proyecto un mayor desempeño de tiempo y un mayor presupuesto.

1.1. Motivación

La herramienta de software newFASANT, en desarrollo desde 1995, ofreciendo una solución que permite realizar simulaciones electromagnéticas de distinto tipo con el fin de satisfacer las necesidades que un ingeniero encargado de diseñar este tipo de sistemas puede necesitar.

Esta solución de software está diseñada de forma modular, ofreciendo distintos componentes que se integran dentro del programa principal. Cuando un cliente adquiere un módulo, se desbloquea y puede construir proyectos de dicho tipo en la interfaz de usuario y posteriormente simularlos para obtener unos resultados que dependen del tipo de proyecto.

La interfaz de usuario de este programa está desarrollada en Java y hace uso de librerías gráficas como Java3D para representar modelos geométricos avanzados, incluyendo soporte para representar geometrías NURBS y para importar y exportar archivos en distintos formatos, incluyendo formatos de intercambio que permiten trabajar junto con otro programa CAD, como DXF, IGES o STEP.

En estos momentos la serie 6.x es la versión más reciente de newFASANT. En el pasado hubo un módulo para una versión más antigua que inicialmente fue planteado

para trabajar con guías de onda pero no llegó a ofrecer buenos resultados por lo que se espera que con este nuevo diseño e implementación se consiga obtener un componente para la versión newFASANT 6 fiable y que proporcione resultados precisos.

1.2. Objetivos

El principal objetivo será agregar a la versión 6.x de newFASANT la capacidad de trabajar con guías de onda, incluyendo la posibilidad de incorporarlas en proyectos de antenas y de realizar simulaciones con ellas con las que obtener resultados sobre acoplamientos, cargas y corrientes.

Para poder desempeñar este objetivo, se han definido una serie de objetivos intermedios y de menor calibre que en conjunto componen la elaboración y el desarrollo de este proyecto:

1. Analizar la situación actual del software newFASANT 6, especialmente en referencia a la interfaz gráfica que utiliza el usuario para interactuar el programa y la forma en la que funciona.

2. Analizar especialmente el comportamiento del módulo dentro del cual se integrará el soporte para guías de onda, y determinar cómo funcionan los componentes con los que interactuará este módulo.

3. Implementación del módulo de guías de onda, partiendo de un diseño que se ajuste al diseño actual del programa y que permita al usuario construir guías de onda o importar guías, agregar puertos de alimentación y obtener resultados.

4. Integrar este módulo dentro de la solución actual, interactuando con el resto de componentes del módulo y comunicándose con el núcleo del programa que efectúe los cálculos.

5. Realización de diversas simulaciones y proyectos de ejemplo y elaboración de un sistema de pruebas que permita validar que el módulo tiene el comportamiento esperado.

Adicionalmente, todos los resultados y todo el plan de trabajo será documentado en la presente memoria, presentando los pasos seguidos dentro del desarrollo de este módulo y el planteamiento de nuevas líneas de desarrollo que pueden surgir a partir de este desarrollo.

1.3. Antecedentes

Tal como se indicó anteriormente, newFASANT lleva en desarrollo desde 2010. Los distintos módulos llevan desarrollándose de forma paralela e integrándose dentro de un mismo programa y actualmente ofrece, de cara al desarrollo de nuevas características, un rico framework dentro del cual integrar nuevos componentes resulta sencillo.

El desarrollo del módulo MOM en la versión newFASANT 6 permite trabajar con antenas de distinto tipo y debido a la relación con el tipo de sistema electromagnético que se está realizando en este proyecto, será donde se integre el sistema una vez se ha desarrollado.

El núcleo MONURBS permite realizar cálculos avanzados usando una configuración inicial como entrada y obteniendo una serie de archivos de salida como resultado. Dentro de este proyecto se aprovechará esta capacidad de cálculo integrando las guías de onda y sus puertos con la entrada que recibe MONURBS y analizando la salida que genera MONURBS.

Como se indicó previamente, hubo un intento anterior por desarrollar un módulo de circuitos capaz de analizar proyectos de guía de onda, sin embargo, su poca estabilidad y su limitado funcionamiento hizo que se buscase desarrollar una mejor implementación que ofrezca mejores resultados.

2 Base teórica

2.1. Guías de onda

Las guías de onda son estructuras utilizadas para transmitir ondas electromagnéticas entre dos puntos del espacio. Son un tipo particular de medio de transmisión utilizado para poder transmitir ondas en frecuencias para las que otros tipos de líneas de transmisión no permiten hacerlo con una eficacia aceptable debido a las pérdidas presentadas.

Las líneas de transmisión son utilizadas para transportar ondas electromagnéticas, por ejemplo para conectar un transmisor con una antena, u otra antena con un receptor. No obstante, las líneas de transmisión tienen como característica que a partir de las frecuencias de la banda UHF (situadas a partir de los 300 MHz), la atenuación provoca unas pérdidas inaceptables a la hora de transmitir ondas de forma satisfactoria.

Esta **atenuación** es el resultado de las pérdidas de potencia ocurridas dentro del medio que transporta la onda y es una característica inevitable en cualquier medio de transmisión que se utilice para portar las ondas. Las guías de onda estudiadas durante esta sección son ventajosas en el rango del espectro electromagnético de las microondas ya que se vuelven óptimas ante frecuencias en las que una línea de transmisión no es capaz. No obstante, es importante tener en cuenta que las guías de onda también sufrirán esta atenuación. Cada tipo de guía de onda que podamos encontrar estará diseñada para ser usada en un rango de frecuencias particular.

Los distintos tipos de guía de onda serán presentados en esta sección junto a un estudio de las propiedades físicas más importantes que caracterizan a las guías de onda y que serán usadas para adquirir un dominio sobre los elementos con los que vamos a trabajar durante este proyecto.

2.1.1. Principio de operación de las guías de onda

Las guías de onda funcionan propagando energía electromagnética por el interior de su estructura, siempre que la estructura sea hueca o contenga un dieléctrico, y siempre que la guía tenga el tamaño adecuado y se encuentre correctamente excitada [2].

Durante esta sección, asumiremos que la onda electromagnética que está siendo guiada por la estructura avanza en la dirección del eje Z. En consecuencia, la propia estructura tubular que define la guía de onda también estará orientada en el mismo eje, de modo que el plano que define la sección de la guía queda definida por los ejes X e Y.

Para una onda electromagnética que avanza en el eje Z, podemos asumir dos componentes esenciales. Una será la componente longitudinal, que es aquella que avanza en la misma dirección que la onda; y otra será la componente transversal, que es aquella que avanza de forma perpendicular a la onda.

Una guía de onda puede ser analizada mediante las ecuaciones de Maxwell con una serie de condiciones de frontera determinadas por las cualidades de la propia guía de onda. La onda que atraviesa esa guía de onda estará definida por la ecuación del campo eléctrico y la ecuación del campo magnético, descritas como:

$$\begin{aligned} \overrightarrow{E}(x,y,z,t) &= \overrightarrow{E}(x,y) \cdot e^{j(\omega t - \beta z)} \\ \overrightarrow{H}(x,y,z,t) &= \overrightarrow{H}(x,y) \cdot e^{j(\omega t - \beta z)} \end{aligned} \tag{2.1}$$

Sin embargo, debido a que esa onda puede ser descompuesta, las ecuaciones del campo eléctrico y magnético también pueden ser descompuestas, considerando por un lado el campo transversal, definido en función de los ejes XY, y el campo longitudinal, definido en función del eje Z:

$$\begin{aligned} \overrightarrow{E}(x,y,z,t) &= \hat{x}\overrightarrow{E_x}(x,y) + \hat{y}\overrightarrow{E_y}(x,y) + \hat{z}\overrightarrow{E_z}(x,y) = \overrightarrow{E_\perp} + \hat{z}\overrightarrow{E_z}(x,y) \\ \overrightarrow{H}(x,y,z,t) &= \hat{x}\overrightarrow{H_x}(x,y) + \hat{y}\overrightarrow{H_y}(x,y) + \hat{z}\overrightarrow{H_z}(x,y) = \overrightarrow{H_\perp} + \hat{z}\overrightarrow{H_z}(x,y) \end{aligned} \tag{2.2}$$

Estas ecuaciones pueden ser aplicadas a las ecuaciones de Maxwell con el objetivo de obtener el conjunto de ecuaciones [3]:

$$\begin{aligned} \nabla_\perp \overrightarrow{E_z} \times \hat{z} - j\beta\hat{z} \times \overrightarrow{E_\perp} &= -j\omega\mu\overrightarrow{H_\perp} \\ \nabla_\perp \overrightarrow{H_z} \times \hat{z} - j\beta\hat{z} \times \overrightarrow{H_\perp} &= -j\omega\mu\overrightarrow{E_\perp} \\ \nabla_\perp \times \overrightarrow{E_\perp} + j\omega\mu\hat{z}\overrightarrow{H_z} &= 0 \\ \nabla_\perp \times \overrightarrow{H_\perp} + j\omega\varepsilon\hat{z}\overrightarrow{E_z} &= 0 \\ \nabla_\perp \cdot \overrightarrow{E_\perp} - j\beta\overrightarrow{E_z} &= 0 \\ \nabla_\perp \cdot \overrightarrow{H_\perp} - j\beta\overrightarrow{H_z} &= 0 \end{aligned} \tag{2.3}$$

La resolución de este conjunto de ecuaciones arroja varias soluciones a las que se les asocian distintos modos de operación. Estos modos definen la configuración que determina de qué forma la energía de la onda es propagada a través de la guía [4].

Los dos modos son el longitudinal y el transversal. En el caso del modo longitudinal la energía se propaga como una onda estacionaria que recorre la guía de onda. Más interesante resulta, no obstante, el modo transversal, que es el que define el comportamiento que tienen las ondas sobre el plano perpendicular a la dirección de propagación de la onda.

2.1.2. Propiedades elementales de las guías de onda

Modos de operación

Dependiendo de cuando las componentes longitudinales (E_z y H_z) tienen un valor igual o distinto a cero, podemos considerar varios modos de operación:

- TEM (Modo Transversal Electromagnético): $E_z = 0, H_z = 0$.
- TE (Modo Transversal Eléctrico): $E_z = 0, H_z \neq 0$.
- TM (modo Transversal Magnético): $E_z \neq 0, H_z = 0$.
- Modo Híbrido: $E_z \neq 0, H_z \neq 0$.

Puesto que las ecuaciones de Maxwell tienen múltiples soluciones, a cada solución se le asocia un modo de operación. Incluso en el caso de modos de operación del mismo tipo (por ejemplo TE frente a TE o TM frente a TM), existen distintos modos de operación debido a la forma en la que las ondas se configuren dentro de la estructura, por lo que un modo de operación también se define con un par de subíndices enteros $m, n \in \mathbb{Z}$, que indican la configuración concreta, por ejemplo, TM_{01} o TE_{10}.

Un estudio más preciso sobre los modos de propagación de la onda depende del tipo de guía de onda particular con el que se esté trabajando, por lo que serán desarrollados en las posteriores secciones, en las que se tratarán algunos tipos particulares de guía de onda que serán los que finalmente se implementen dentro del módulo desarrollado en este trabajo.

Frecuencia de corte

En guías de onda, se considera la **frecuencia de corte** a la frecuencia a partir de la cual la guía de onda es capaz de propagar en un determinado modo de propagación.

No debe confundirse la frecuencia de corte de una guía de onda con la frecuencia de onda de otro medio de transmisión tal como una línea de transmisión o un filtro. En ambos tipos de sistemas RF existen dos conceptos conocidos como frecuencia de corte, pero su definición es distinta. En esos sistemas, la frecuencia de corte se corresponde con aquella a partir de la cual la atenuación producida en el medio disipa la potencia de la onda de modo que la hace poco práctica de cara a la transmisión de datos utilizando ondas electromagnéticas debido a que se pierde la información contenida en las ondas.

En el caso de las guías de onda no obstante, las frecuencias de corte miden frecuencias mínimas. Por debajo de dicha frecuencia, la guía de onda no podrá transmitir la onda proporcionada.

La frecuencia de corte particular para una estructura depende de su forma geométrica, de sus dimensiones y de su modo de propagación, por lo que un estudio de la frecuencia de corte para los tipos de guía de onda que se analizarán en esta sección se presentará junto al resto de tipos de guía de onda.

Las guías de onda tienen una **frecuencia de corte inferior**, por debajo de la cual una onda electromagnética no propaga adecuadamente. Para comprender por qué una guía de onda requiere una frecuencia mínima para propagar, se puede observar que cuanto más baja sea una frecuencia, mayor es su longitud de onda en tanto que son inversamente proporcionales. Si una onda tiene una longitud lo suficientemente grande, eventualmente la longitud de la onda será mayor que las propias dimensiones de la sección de la guía de onda por lo que no podrá propagar correctamente en modo transversal.

No obstante, las guías también tienen una **frecuencia de corte superior**, a partir de la cual no es posible utilizarla en un modo único de propagación. Alcanzada esa frecuencia de corte, otros modos de la guía de onda comenzarán a propagar debido a que se habrá alcanzado la frecuencia de corte para ese modo. Cuando se trabaja con guías multimodo esto no es un problema, pero en la mayoría de casos lo que se busca es trabajar con guías en un único modo. Si una onda tuviese la frecuencia suficiente como para que se encuentre por encima de la frecuencia de corte inferior de varios modos de propagación, no se podría determinar con seguridad en qué modo de propagación está funcionando una guía en un momento determinado [3]. En consecuencia, existiría más de un modo funcionando a la vez y eso podría provocar distorsiones que afecten a la calidad de la señal transmitida por la guía de onda.

Dada una guía de onda y una serie de modos de operación asociados con una frecuencia inferior de corte, existe un **modo dominante**, que es el modo para el que se diseña, y que coincide con el modo de operación más pequeño, es decir, aquél que tiene la menor frecuencia de corte de todas. Este modo es el modo en el que se debe usar la guía de onda porque garantiza que al tener la menor frecuencia de corte no habrá otro modo de operación con una frecuencia de corte menor que pueda transmitir a la vez.

A partir de dicho modo dominante, se establece la frecuencia de corte superior, como la frecuencia de corte inferior del modo de propagación inmediatamente superior a dicho modo dominante. A partir de esa frecuencia de corte superior, dos modos podrían operar de forma simultánea por lo que no es seguro efectuar transmisiones.

El espectro comprendido entre la frecuencia de corte inferior y la frecuencia de corte superior es considerado como **rango de operación**, y es un valor que el fabricante de una guía de onda indica de modo que no se use ni por debajo (porque no habría ningún modo de propagación que pudiese propagar al estar por debajo de todas las frecuencias de corte) ni por encima (porque habría más de un modo de operación por encima de su frecuencia de corte correspondiente).

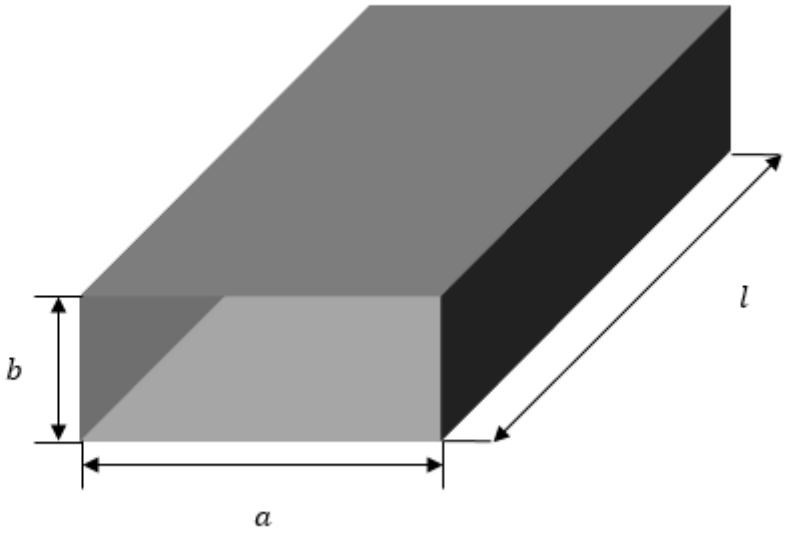


Figura 2.1: Guía de onda rectangular

Pérdidas de energía

Todos los cálculos presentados en esta sección asumen una guía de onda ideal sin pérdidas. En la realidad, lo normal es que las guías de onda provoquen una serie de pérdidas debido a una atenuación propia de la guía de onda inevitable [5].

Esto también tiene como consecuencia que en las frecuencias más próximas a las frecuencias de corte pueden producirse pérdidas significativas, por lo que en general no se utilizará todo el ancho de banda de la guía de onda para hacerla operar, si no que se trabajará con una frecuencia mínima y una máxima que está próxima a las frecuencias de corte inferior y superior [2].

2.1.3. Guías de onda rectangulares

Las guías de onda rectangulares son un tipo de guía de onda donde la estructura tiene forma de tubo, las paredes que conforman el recubrimiento son conductoras, y el interior de la guía de onda, o bien está hueco y está relleno con aire, o contiene un dieléctrico.

La sección de esta guía de onda se caracteriza por tener una forma rectangular compuesta de dos dimensiones, comúnmente denominadas *a* y *b*, tal como puede apreciarse en la figura 2.1. *a* hace referencia al lado largo de la sección y *b* hace referencia al lado corto.

Análisis de las guías de onda rectangulares

En una guía de onda rectangular, el modo de propagación principal es el transversal eléctrico (TE_{mn}), donde m y n son dos enteros que cumplen que $m \geq 0, n \geq 0$ con la particularidad de que debemos excluir el caso en el que $m = n = 0$.

m representa la cantidad de medias ondas que son capaces de penetrar de forma transversal a la dirección de propagación de la onda en la guía en la dirección de a, mientras que n representará la cantidad de medias en la dirección de b. En función de los valores de m y n el aspecto de los campos transversales a la onda varía.

El modo dominante para una guía de onda rectangular es el modo TE_{10}. Se trata del modo con la menor frecuencia de corte por lo que la resolución de problemas está bastante simplificada.

En el caso de guías de onda rectangulares, la frecuencia de corte depende de los valores de m y n establecidos, a través de la siguiente ecuación:

$$f_c = c\sqrt{(\frac{m}{2a})^2 + (\frac{n}{2b})^2} \tag{2.4}$$

En el caso del modo de propagación TE_{10}, el principal dentro de una guía de onda rectangular, la frecuencia de corte puede calcularse simplemente como:

$$f_c = \frac{c}{2a} \tag{2.5}$$

En ambos casos, c es la velocidad de la luz en el vacío.

Por otro lado, conocida la frecuencia de corte también resulta particularmente interesante considerar la longitud de onda de corte de la guía de onda, obtenida como:

$$\lambda_c = \frac{c}{f_c} = \frac{1}{\sqrt{(\frac{m}{2a})^2 + (\frac{n}{2b})^2}} \tag{2.6}$$

Además, no debemos olvidar que tal como se indicó en la sección anterior todas las guías de onda tienen una frecuencia de corte superior que se corresponde con la frecuencia de corte inferior del modo de propagación inmediatamente superior al modo para el que se diseñó la guía de onda.

En el caso de guías de onda rectangulares, diseñadas para funcionar en modo TE_{10}, hay que localizar el siguiente modo de operación. El modo de operación inmediatamente superior puede ser o TE_{01} o TE_{20}, dependiendo de cuál de los dos tenga la menor frecuencia de corte inferior. Esa frecuencia de corte inferior actuará como frecuencia de corte superior para el modo TE_{10}. Estaremos interesados en las situaciones en las que la frecuencia de corte para TE_{20} sea la menor, debido a que nos proporcionará un mayor ancho de banda, en tanto que $f_{20} = 2f_{10}$.

Puesto que la frecuencia de corte depende de las dimensiones de la guía de onda, estaremos interesados en obtener la proporción apropiada que permita que eso se cumpla. Si planteamos una inecuación, podemos determinar esa proporción.

$$\begin{aligned} f_{20} \leq f_{01} \Rightarrow \lambda_{20} &\geq \lambda_{01} \\ a &\geq 2b \\ \frac{a}{2} &\geq b \end{aligned} \tag{2.7}$$

Como consecuencia, si una guía de onda cumple una proporción tal que $b \leq a/2$, el modo superior al dominante será TE_{20}, lo que permitirá maximizar el ancho de banda posible utilizado en la guía de onda. En cambio, para el resto de situaciones, la frecuencia de corte se situará en la del modo de operación TE_{01}.

En definitiva, para una frecuencia f, es necesario proporcionar una guía de onda con unas dimensiones que permita que se cumpla la siguiente condición:

$$\begin{cases} \frac{c}{2a} < f < \frac{c}{a} & \text{si } b \leq \frac{a}{2} \\ \frac{c}{2a} < f < \frac{c}{2b} & \text{si } \frac{a}{2} < b < a \end{cases} \tag{2.8}$$

Análisis particular de las pérdidas en guías de onda rectangulares

En el análisis general de las guías de onda, se indicó que las guías de onda tienen asociadas unas pérdidas debido al propio medio, las cuales pueden provocar unos niveles de atenuación significativos en las frecuencias próximas a las de corte inferior y superior.

En el caso de las guías de onda rectangulares se suele trabajar con una frecuencia mínima correspondiente al 125 % de la frecuencia de corte inferior de la guía de onda [5]. Para la frecuencia máxima, cuando se cumple que $b \leq a/2$, se utiliza un 189 % la frecuencia de corte inferior, un 94,5 % la frecuencia de corte superior.

Por proponer un mero ejemplo demostrativo de cómo se efectúan estos cálculos, cuando tratamos con una guía de onda con dimensiones $a = 0{,}02286$ y $b = 0{,}01086$, la frecuencia de corte inferior se calcularía como:

$$f_{10} = \frac{c}{2a} = \frac{3 \cdot 10^8}{2 \cdot 0{,}02286} = 6{,}561 \cdot 10^9 \tag{2.9}$$

Mientras que la frecuencia de corte superior, al ser $b \leq a/2$, se calcula como:

$$f_{20} = 2f_{10} = 1{,}312 \cdot 10^{10} \tag{2.10}$$

El rango de frecuencias aceptable no estará entre $6{,}56 \cdot 10^9$ y $1{,}31 \cdot 10^{10}$ porque cuando nos acerquemos a las frecuencias de corte podrían producirse significativas

pérdidas. En su lugar se utilizan como mínimo y máximo el 125 % y el 189 % de la frecuencia de corte inferior.

$$\begin{aligned} f_{min} &= 1{,}25 \cdot 6{,}561 \cdot 10^9 = 8{,}20 \cdot 10^9 \\ f_{max} &= 1{,}89 \cdot 6{,}561 \cdot 10^9 = 1{,}24 \cdot 10^1 0 \end{aligned} \tag{2.11}$$

Dimensiones de las guías de onda

Como ha quedado claro por este análisis, las propiedades de la guía de onda, tales como la frecuencia de corte o el rango de frecuencias para la que una guía de onda está diseñada, dependen de la geometría de la guía de onda y sobre todo, de los valores a y b que definen las dimensiones de la sección de la guía de onda.

La industria ha diseñado unos estándares para trabajar con guías de onda rectangulares mediante la cual se le asigna a algunas guías de onda con dimensiones particulares unos nombres que hagan más fácil su identificación. Bajo este estándar, una guía de onda con un identificador especial tiene siempre las mismas dimensiones de a y b, lo que hace que universalmente sea reconocida tanto las dimensiones como el rango de frecuencias para el que está diseñada la guía de onda.

El principal estándar empleado es el WR, propuesto por la EIA (Electronic Industries Alliance) y utilizado en la mayoría de partes del mundo. Este estándar está propuesto por Estados Unidos y en consecuencia tiene algunas particularidades, como el hecho de que todas las unidades de a y b se expresen en pulgadas. Todas las ecuaciones presentadas están expresadas en términos de unidades del sistema internacional (metros), por lo que será necesario aplicar las conversiones necesarias.

Además del estándar de la EIA, existen otros dos estándares menos empleados.

- WG el estándar propuesto por la RCSC (Radio Components Standardization Committee), usado principalmente en el Reino Unido. Todas las guías de onda WR tienen un equivalente en WG.
- R es el estándar es el propuesto por la IEC (International Electrotechnical Commission). No todas las guías de onda presentes en los estándares WR y WC tienen equivalente en el sistema R.

Durante el desarrollo de este módulo, trabajaremos con el sistema WR al ser el más aceptado internacionalmente.

El estándar WR establece unos identificadores ante las guías de onda que se compone agregando el prefijo WR y después indicando un número que se compone multiplicando el valor de a por 100 expresado en pulgadas.

Por ejemplo, sea una guía de onda rectangular cuyas dimensiones en pulgadas son $0,900 \cdot 0,400$. Esta guía de onda será la WR-90 debido a que $0,900 \cdot 100 = 90$. En la tabla 2.1 se presentan las principales guías de onda que han recibido nombres por parte de la EIA, incluyendo dimensiones en pulgadas, milímetros, y el rango de frecuencias para el que la guía está preparada para funcionar.

Tabla 2.1: Dimensiones estandar para guías de onda

Nombre	Dimensiones (mm)	Dimensiones (in)	Rango (GHz)
WR340	$86{,}36 \times 43{,}18$	$3{,}400 \times 1{,}700$	$2{,}20 - 3{,}30$
WR284	$72{,}136 \times 34{,}036$	$2{,}840 \times 1{,}340$	$2{,}60 - 3{,}95$
WR229	$58{,}166 \times 29{,}21$	$2{,}290 \times 1{,}150$	$3{,}30 - 4{,}90$
WR187	$47{,}5488 \times 22{,}1488$	$1{,}872 \times 0{,}872$	$3{,}95 - 5{,}85$
WR159	$40{,}386 \times 20{,}193$	$1{,}590 \times 0{,}795$	$4{,}90 - 7{,}05$
WR137	$34{,}8488 \times 15{,}7988$	$1{,}372 \times 0{,}622$	$5{,}85 - 8{,}20$
WR112	$28{,}4988 \times 12{,}6238$	$1{,}122 \times 0{,}497$	$7{,}05 - 10{,}00$
WR90	$22{,}86 \times 10{,}16$	$0{,}900 \times 0{,}400$	$8{,}2 - 12{,}4$
WR75	$19{,}05 \times 9{,}525$	$0{,}750 \times 0{,}375$	$10{,}0 - 15{,}0$
WR62	$15{,}7988 \times 7{,}8994$	$0{,}622 \times 0{,}311$	$12{,}4 - 18{,}0$
WR51	$12{,}954 \times 6{,}477$	$0{,}510 \times 0{,}255$	$15{,}0 - 22{,}0$
WR42	$10{,}668 \times 4{,}318$	$0{,}420 \times 0{,}170$	$18{,}0 - 26{,}5$
WR28	$7{,}112 \times 3{,}556$	$0{,}280 \times 0{,}140$	$26{,}5 - 40{,}0$
WR22	$5{,}6896 \times 2{,}8448$	$0{,}224 \times 0{,}112$	$33{,}0 - 50{,}0$
WR19	$4{,}7752 \times 2{,}3876$	$0{,}188 \times 0{,}094$	$40{,}0 - 60{,}0$
WR15	$3{,}7592 \times 1{,}8796$	$0{,}148 \times 0{,}074$	$50{,}0 - 75{,}0$
WR12	$3{,}0988 \times 1{,}5494$	$0{,}122 \times 0{,}061$	$60{,}0 - 90{,}0$

2.1.4. Guías de onda circulares

El otro tipo de guía de onda predominante es la guía de onda circular. Una guía de onda circular es aquella que tiene una sección circular de radio r. En la figura 2.2 se puede ver un ejemplo de guía de onda circular.

Análisis de las guías de onda circulares

En guías de onda circulares también existen modos de operación transversales eléctricos (TE) y transversales magnéticos (TM), aunque los subíndices que usamos para nombrar los modos son distintos.

En el caso de una guía de onda circular que propague en modo TE_{mn} o TM_{mn}, el primer subíndice, m, es utilizado para indicar el número de ondas concéntricas que hay en el plano transversal a la dirección de propagación de la guía de onda, mientras que n hace referencia al número de mitades de guías de onda que hay en torno al diámetro [6].

En el caso de las guías de onda circulares, el modo de propagación dominante es el TE_{11}. No obstante, como se tendrá ocasión de comprobar más adelante, no resulta tan directo determinar las frecuencias de corte en este caso.

En un modo de propagación transversal eléctrico TE_{mn}, la frecuencia de corte de la guía de onda se define a partir de la ecuación:

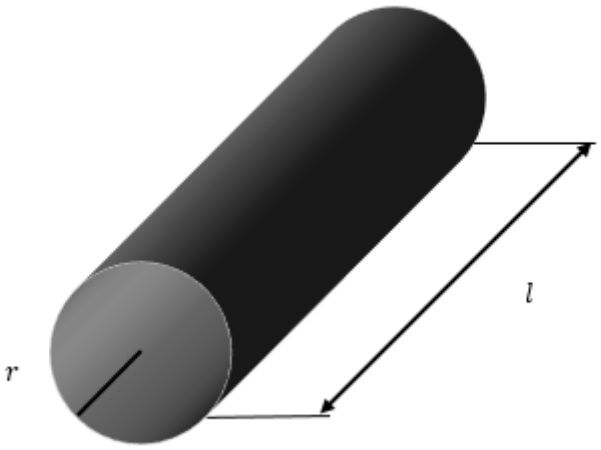


Figura 2.2: Guía de onda circular

Tabla 2.2: Coeficientes χ'_{ij} obtenidos para M=0..3 y N=1..3

	M=0	M=1	M=2	M=3
N=1	3.8318	1.8412	3.0542	4.2012
N=2	7.0156	5.3315	6.7062	8.0153
N=3	10.1735	8.5363	9.9695	

$$f_c = \frac{\chi'_{mn} c}{2\pi r} \tag{2.12}$$

χ'_{mn} es un coeficiente que depende de los valores de m y de n y que se calcula a partir de la raíz n de la derivada función de Bessel de primer tipo $J_m(r)$. La derivada se calcula a partir de la función de Bessel original, definida como:

$$J_\alpha(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\Gamma(m+\alpha+1)} (\frac{x}{2})^{2m+\alpha} \tag{2.13}$$

Para algunos valores de m y n, en la tabla 2.2 se presentan los resultados de χ'_{mn} [7].

Cuando lo aplicamos al modo de propagación dominante, TE_{11}, obtenemos que la frecuencia de corte se determina como:

$$f_c = \frac{1{,}8412 c}{2\pi r} \tag{2.14}$$

El siguiente modo de propagación superior al TE_{11} es el TM_{01}. Esto es debido a que en el caso del modo TM, la frecuencia de corte se determina mediante χ_{mn}, no χ'_{mn}, como coeficiente, por lo que tenemos que atender a la función de Bessel sin derivar. En ese caso, algunos de los coeficientes se presentan en la tabla 2.3 [7].

Tabla 2.3: Coeficientes χ_{ij} obtenidos para M=0..3 y N=1..3

	M=0	M=1	M=2	M=3
N=1	2.405	3.832	5.136	6.380
N=2	5.520	7.016	8.417	9.761
N=3	8.654	10.173	11.620	13.015

Para modos TM, la frecuencia de corte se calcula como:

$$f_c = \frac{\chi_{mn} c}{2\pi r} \tag{2.15}$$

A la vista de esto, el modo inmediatamente superior al modo dominante TE_{11} es el TM_{01}, ya que su coeficiente, 2.405, es el inmediatamente mayor a 1.8412. La frecuencia de corte para el modo TM_{01} se calcula como:

$$f_c = \frac{2{,}405 c}{2\pi r} \tag{2.16}$$

Al igual que ocurre con las guías de onda rectangulares, estamos interesados en utilizar guías de onda que permitan transmitir ondas con una frecuencia que se comprenda entre la frecuencia de corte TE_{11} y la TM_{01}. Es decir, que para poder transmitir una onda con una frecuencia f en una guía de onda con un radio r, deberá cumplirse lo siguiente:

$$\frac{1{,}8412 c}{2\pi r} < f < \frac{2{,}405 c}{2\pi r} \tag{2.17}$$

2.2. Parámetros de dispersión

Durante el estudio de sistemas de transmisión basados en microondas, interesa obtener los distintos parámetros presentes en un circuito de microondas, tal como el que utiliza guías de onda para propagar ondas electromagnéticas.

El módulo que se desarrolla en este trabajo de fin de grado busca el estudio de estos parámetros con el objetivo de que el ingeniero que está utilizando el software pueda, a partir de la simulación de su proyecto, determinar por medio de los parámetros obtenidos los resultados que necesite. Para ofrecer estas funcionalidades en el módulo será necesario previamente obtener una base teórica de estas propiedades a fin de conocer lo que se pretende desarrollar.

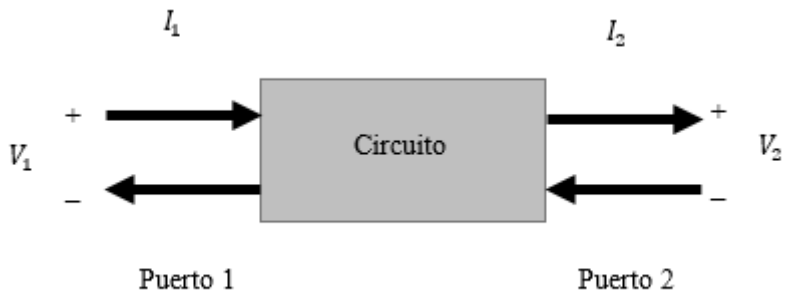


Figura 2.3: Representación de un sistema de dos puertos.

2.2.1. Circuitos y sistemas de múltiples puertos

En el área de las microondas un **circuito** es una abstracción que permite representar un sistema de transmisión basado en microondas como una caja negra, ocultando los componentes internos y manteniendo únicamente los puntos terminales que permiten conectar al sistema con el exterior, denominados **puertos** [8].

La ventaja de trabajar con circuitos y puertos es que permite reducir la complejidad de estos sistemas y trabajar en función de una serie de señales de entrada y de salida. Por ejemplo, una onda que entra a un circuito a través de un puerto constituye una entrada y una onda que sale de un circuito por otro circuito constituye una salida.

Un circuito puede tener un número variable de puertos, aunque en este caso trabajaremos con sistemas de dos puertos por simplificar bastante todas las operaciones que se hagan dentro de este módulo. En el caso de que los proyectos construidos por el usuario tengan más puertos, se permutarán pares de puertos para trabajar con sistemas de dos puertos.

Un sistema de dos puertos también es conocido en literatura como redes de cuatro terminales, debido a que un puerto contiene dos terminales [9]. Un terminal será de entrada, y servirá para hacer entrar energía en el circuito, mientras que otro será el terminal de salida y servirá para dejar salir energía del circuito. Se identifican por unas flechas situadas junto a los terminales indicando la dirección del recorrido de la energía. En la figura 2.3 se puede visualizar un ejemplo de este sistema de dos puertos.

Podemos realizar un análisis de un sistema de dos puertos siempre que sea lineal, es decir, que haya una relación entre los voltajes y las intensidades de dicho circuito. El factor que relaciona el voltaje con la intensidad se denominará impedancia y es lineal porque la siguiente igualdad se cumple:

$$V = IZ \tag{2.18}$$

2.2.2. Parámetros de un sistema de dos puertos lineal

A partir de un sistema de dos puertos lineal podemos obtener varios parámetros. Los principales parámetros de un circuito que vamos a estudiar en esta sección y que luego serán obtenidos con el programa son:

- La matriz de impedancias, caracterizada por los parámetros Z.
- La matriz de admitancias, compuesta por los parámetros Y.
- Los parámetros de dispersión, conocidos también como parámetros S.

2.2.3. Parámetros Z

Los parámetros Z representan la relación entre los voltajes y las corrientes. Sea V_1 el voltaje en el puerto 1, V_2 el voltaje en el puerto 2, I_1 la intensidad en el puerto 1 y I_2 la intensidad en el puerto 2, los parámetros Z pueden calcularse a partir de la resolución del siguiente sistema matricial [3]:

$$\begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} Z_{11} & Z_{12} \\ Z_{21} & Z_{22} \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \end{bmatrix} \tag{2.19}$$

Cada elemento Z_{ij} será un número complejo. Este sistema matricial también puede determinarse calculando de forma independiente cada elemento Z_{ij}, algo que se puede hacer excitando el puerto j aplicándole una intensidad I_j, y después midiendo el voltaje en el puerto i, que al igual que el resto de puertos que no sean el propio puerto j estarán abiertos. Es decir:

$$Z_{ij} = \left.\frac{V_i}{I_j}\right|_{I_k=0 \forall k \neq j} \tag{2.20}$$

Es necesario considerar algunos casos particulares que deben cumplirse ante redes de puertos concretas.

Cuando una red es recíproca, debe cumplirse que la matriz de impedancias sea simétrica [8]. Una red es recíproca si sólo contiene elementos pasivos, como resistores o capacitores. En consecuencia de esto, si en el puerto 1 entra un voltaje de entrada V_I, en el puerto 2 se obtiene un voltaje de salida V_O, y se cumplirá que de forma recíproca, cuando en el puerto 2 entre el mismo voltaje de entrada V_I, en el puerto 1 saldrá de nuevo el voltaje V_O.

Si además se trata de una red recíproca sin pérdidas, es decir, una red en la que no haya elementos que disipen energía, entonces los parámetros Z obtenidos, que recordemos son números complejos, sólo tendrán parte imaginaria.

2.2.4. Parámetros Y

Los parámetros Y permiten determinar la matriz de admitancia, que permite relacionar las corrientes con los voltajes en un sistema de dos puertos. La matriz de admitancia es la inversa de la matriz de impedancias, de modo que se puede determinar resolviendo el sistema matricial:

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \tag{2.21}$$

O, si los Z parámetros ya han sido calculados, como:

$$[Y] = [Z]^{-1} \tag{2.22}$$

El parámetro Y_{ij} también puede ser determinado de forma experimental determinando cuál es la intensidad de corriente obtenida en el puerto i cuando todos los puertos están cortocircuitados (sin voltaje) a excepción del puerto j por donde está entrando un voltaje V_j, es decir:

$$Y_{ij} = \left.\frac{I_i}{V_j}\right|_{V_k=0 \forall k \neq j} \tag{2.23}$$

La matriz de admitancia cumple las mismas propiedades especiales que la de impedancias. En particular:

- Cuando el circuito es recíproco, la matriz obtenida es simétrica.
- Cuando el circuito no tiene pérdidas, todos los valores obtenidos son imaginarios.

2.2.5. Parámetros S

Los parámetros de dispersión o parámetros S expresan una relación más avanzada. En este caso lo que se busca es la relación entre las ondas reflejadas y las ondas incidentes. La onda incidente es caracterizada como el voltaje de entrada en un puerto, mientras que la onda reflejada es el voltaje de salida de un puerto.

En la figura 2.4 se presentan esos parámetros en un sistema de dos puertos. Las ondas incidentes son representadas como a_i donde i es el puerto por el que entran. Las ondas reflejadas se representan en su lugar como b_i siendo i el puerto reflejado.

Los valores de a_i y b_i se determinan a partir de los voltajes, las intensidades, y una impedancia característica referida como Z_0 [8]. Esta impedancia característica es la que se obtiene cuando el medio de transmisión es uniforme en todos los puntos

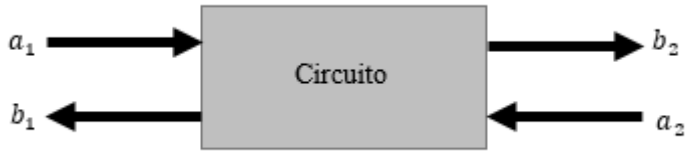


Figura 2.4: Representación de los voltajes incidentes y reflejados en un sistema de dos puertos.

de su longitud. Para determinar la onda incidente y la onda reflejada en un puerto i necesitamos identificar también el voltaje en el mismo puerto y la intensidad en el mismo puerto. a es calculado como:

$$a = \frac{V + Z_0 I}{2\sqrt{Z_0}} \tag{2.24}$$

Mientras que b es calculado como:

$$b = \frac{V - Z_0 I}{2\sqrt{Z_0}} \tag{2.25}$$

En líneas generales, la resolución del sistema matricial que permite determinar la matriz de parámetros S es:

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} S_{11} & S_{12} \\ S_{21} & S_{22} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{2.26}$$

Experimentalmente, se pueden calcular los parámetros S_{ij} de una forma parecida a como hemos visto con los Y parámetros y con los Z parámetros. El parámetro S_{ij} se determina examinando el voltaje reflejado obtenido en el puerto i cuando en el puerto j se aplica un voltaje incidente. Sin embargo, a diferencia de otros parámetros anteriormente vistos, en este caso no se cortocircuitan los puntos si no que se introducen unas cargas equivalentes de modo que su resistencia sea igual a la resistencia característica del circuito (Z_0). Con esto se previene la reflexión de las ondas incidentes sobre puertos en los que no nos estamos centrando. Los valores pueden determinarse experimentalmente mediante el cálculo:

$$S_{ij} = \left.\frac{b_i}{a_j}\right|_{V_k=0 \forall k \neq j} \tag{2.27}$$

Existen dos tipos de parámetros de dispersión [3]:

- Los coeficientes de reflexión son aquellos coeficientes S_{ij} en los que $i = j$. Permiten comprobar, para un puerto sobre el que incide una onda, cuánto de ese voltaje es reflejado sobre el mismo puerto.

- Los coeficientes de transmisión en cambio son los coeficientes S_{ij} donde $i \neq j$. En estos coeficientes vemos la onda reflejada sobre otro puerto distinto a aquél en el que incide la onda, por lo que comprueba cuánto de ese voltaje es transmitido.

3 Estado del arte

3.1. Herramientas de simulación electromagnética

Las herramientas de simulación electromagnética son sistemas de software que utilizan las técnicas de electromagnetismo computacional para ponerlas al servicio de los usuarios finales. Por medio de las herramientas de simulación electromagnética un ingeniero puede realizar en un ordenador simulaciones efectivas de situaciones en las que componentes electromagnéticos se ven involucrados, como por ejemplo sistemas RF, sistemas de arrays reflectantes o radomos, de una forma cómoda y en la mayoría de los casos amigable.

El usuario normalmente tiene que aportar información sobre el proyecto que está intentando llevar a cabo, haciendo algunos ajustes, importando o construyendo una representación del entorno, y aportando los parámetros de las antenas u otros mecanismos electromagnéticos que van a verse involucrados en la simulación. La representación del entorno consiste en la disposición de los elementos físicos que forman parte de la simulación (como por ejemplo un avión, una guía de onda, un submarino...), y deberá ser construida por el usuario si el programa que está utilizando dispone de esa funcionalidad; o, en la mayoría de los casos, importándola una vez ha sido generada con otros programas de diseño CAD más avanzados.

Una vez el usuario tiene toda la información que va a necesitar en el proyecto construido por medio de la interfaz, tendrá que ejecutar la simulación, realizando los cálculos, para lo que se utilizan solvers especializados, como se presentará en la sección 3.1.1. Los solvers recogen los datos del proyecto y generan una serie de resultados que pueden ser estudiados. Algunos de estos resultados se componen exclusivamente de números, pero lo más común es que, debido al funcionamiento de un solver, sea necesario interpretar de forma gráfica los resultados, para visualizar efectos de campos, gráficas e incluso en algunas situaciones diagramas representados sobre las propias geometrías.

3.1.1. Solvers

Los solvers son los programas principales dentro de una herramienta de simulación electromagnética ya que es quien realiza todas las operaciones matemáticas y en según

que casos físicas que hay detrás de cualquier proyecto electromagnético que se realice con la ayuda de herramientas informáticas.

La motivación detrás de la existencia de un solver se encuentra en las ecuaciones de Maxwell, las cuales forman parte de cualquier tipo de problema electromagnético, incluyendo los de guías de onda como se pudo presentar en el capítulo anterior. Solucionar un sistema de ecuaciones de Maxwell es un proceso complejo. Por lo tanto, lo normal es que a la hora de resolver uno de estos sistemas no se trabaje completamente con las ecuaciones de Maxwell, si no que se usen aproximaciones [10].

El **electromagnetismo computacional** es un campo de la ciencia donde se investigan y se desarrollan algoritmos que faciliten las aproximaciones de problemas de electromagnetismo, buscando formas de hacerlos eficientes y con un consumo de recursos, tanto de tiempo, como de memoria, aceptable y optimizable.

Hay que tener en cuenta que el electromagnetismo computacional es un campo multidisciplinar que no puede ser clasificado simplemente como un producto de las ciencias de la computación o como algo que se deba asociar únicamente a la física electromagnética. Lo común es que se dependa también de una base matemática y de conocimientos geométricos que permitan extender y aplicar el potencial del electromagnetismo computacional a las representaciones fieles del mundo que fueron presentadas anteriormente.

Los principales solvers disponibles en el mercado utilizan técnicas CEM que aproximan las ecuaciones de Maxwell de una forma numérica. Debido a que la forma de las geometrías utilizadas tienen relación con los resultados obtenidos, hay un proceso de **discretización** previo a la ejecución de los cálculos. Esta discretización consiste en la división de las geometrías que conforman un proyecto en elementos más pequeños denominados parches que permitan obtener una malla a partir de la geometría, de modo que se pueda obtener una representación mucho más precisa de los puntos que conforman la geometría. La discretización también es un proceso denominado **mallado** debido a dicha generación de mallas.

En cuanto a la obtención de los resultados como tal, es importante conocer los distintos métodos que pueden ser usados para realizar los cálculos, ya que hay que elegir siempre un método apropiado al tipo de problema que se esté intentando desarrollar, con el fin de prevenir errores. Los solvers se encuentran repartidos en dos grupos principales: los que utilizan formas diferenciales de las ecuaciones de Maxwell, y los que utilizan ecuaciones integrales [11].

En los solvers que usan ecuaciones integrales, se trabaja con sistemas lineales avanzados. La tendencia actual es a buscar optimizaciones que ayuden a construir solvers más eficientes aprovechándose de técnicas modernas de computación como el multiprocesamiento, los procesadores vectoriales, o en última instancia, la supercomputación, mediante el empleo de superordenadores y estaciones de trabajo con un número elevado de procesadores y una gran cantidad de memoria RAM en comparación con las prestaciones de los habituales equipos domésticos y de oficina disponibles en la actualidad.

El **método de los momentos** es uno de los solvers integrales más empleados en este momento y el solver con el que se trabajará en posteriores secciones de esta memoria [10]. Conocido como MoM, para poder utilizar este sistema se trabaja con corrientes que fluyen a través de superficies que han sido malladas y convertidas en pequeños parches, y analizando la interacción entre los distintos parches por medio de una matriz que relaciona la interacción de todos los parches.

Resolver dicho sistema matricial no es una operación trivial. De hecho, debido a que se analizan las relaciones entre los distintos parches que conforman las superficies, a menudo se trabaja con matrices con un órden que supera el millar, considerando además que las corrientes son números complejos. No obstante, MoM resulta efectivo en una gran variedad de tipos de proyecto debido al tratamiento que hace de las superficies y debido a que depende de las corrientes en vez de otras propiedades.

Por otro lado, además de los solvers que usan ecuaciones integrales, existen otros que usan formas diferenciales. Uno de los principales algoritmos diferenciales es el **Método del Elemento Finito** (FEM) y es usado por los solvers de algunas de las herramientas de simulación EM presentes en el mercado que se presentan a continuación.

3.1.2. Herramientas EM disponibles en el mercado

Entre las herramientas de simulación actuales estaremos interesadas en aquellas que permiten trabajar con proyectos de guías de ondas, ya que ese tipo de proyectos conforma el ámbito de este trabajo de fin de grado. Se destacan las siguientes alternativas.

- CST Microwave Studio: se trata de una herramienta de software desarrollada por la empresa alemana CST orientada a la simulación de componentes electromagnéticos de alta frecuencia [12]. Dispone de varios *solvers* ocupados de calcular distintas propiedades a partir de proyectos de microondas.
- HFSS Studio: HFSS Studio es un software electromagnético que permite realizar cálculos de campos electromagnéticos en proyectos de altas frecuencias [13]. Entre las propiedades que puede analizar el usuario a partir del cálculo por medio de solvers que usan el método de los elementos finitos están los parámetros de dispersión y el análisis del campo cercano y campo lejano.
- FEKO: La suite de FEKO ofrece un módulo de análisis RF que permite realizar análisis de sistemas de microondas utilizando solvers basados en el método de los momentos y en el método de los elementos finitos [14].
- newFASANT 5: Se trata de una versión anterior del software newFASANT. Como se indica más adelante, esta versión está fuera de soporte y la nueva versión no dispone de esta funcionalidad, que es parte del objetivo de este trabajo de fin de grado. Utiliza el método de los momentos para realizar cálculos sobre circuitos RF, incluyendo microondas, y permite realizar cálculos de los parámetros de dispersión en circuitos microstrip y basados en antenas [15].

3.2. La herramienta de software newFASANT

newFASANT es una herramienta de software desarrollada por el Grupo de Investigación de Electromagnetismo Computacional de la Universidad de Alcalá. Se trata de un software que permite realizar simulaciones electromagnéticas utilizando distintas técnicas de electromagnetismo computacional.

El software está concebido para proporcionar a los usuarios una suite de módulos destinados a resolver problemas electromagnéticos de muy diversos tipos. Cada uno de estos módulos permite realizar un tipo de operaciones, por ejemplo, realizar análisis y diseños de sistemas de antenas, obteniendo distintos parámetros sobre ellas, como la distribución de cargas o el acoplamiento entre distintas antenas; diseño y análisis de sistemas RCS, Chaff...

La solución que ofrece el software de newFASANT puede desglosarse en dos principales componentes: una interfaz de usuario realizando la labor de *front-end* y un conjunto de núcleos que se ocupan del *back-end*.

La interfaz de usuario es la capa que interactúa con el usuario. Inicialmente desarrollado en FORTRAN, fue convertida a Java y por medio de la librería gráfica Java3D permite al usuario construir proyectos, definir las opciones y agregar las distintas geometrías que definen al proyecto. El usuario tiene a su disposición una serie de primitivas que puede agregar al programa, como cajas, esferas o cilindros; aunque también tiene la posibilidad de importar directamente sobre el universo simulado otros objetos importados de otros programas CAD presentes en el mercado en distintos formatos de intercambio como DXF, IGES o STEP.

Además de definir las geometrías del escenario el usuario deberá agregar otras geometrías propias de cada tipo de proyecto. Puesto que el desarrollo de esta memoria está enfocado al módulo de guías de onda tomaremos como ejemplo el módulo MOM. El usuario puede agregar distintas antenas y asociarlas a geometrías introducidas en la interfaz de usuario. Para cada tipo de antena podrá definir algunos parámetros sobre ella.

Cuando el usuario haya definido correctamente su escenario y esté listo para realizar la simulación para obtener los resultados, puede utilizar el menú de simulación para iniciarla. Es ahí donde la interfaz de usuario se comunicará con el *back-end*, el cual está integrado por una serie de núcleos. El núcleo es el programa que recibe los datos de la simulación del usuario, los procesa y finalmente genera unos resultados. Cada tipo de proyecto está asociado con un núcleo, sin perjuicio de que ese núcleo pueda estar siendo usado por varios tipos de proyecto. Lo importante es que el núcleo es el que realiza la parte principal de los cálculos, que no se realizan en Java. Estos núcleos están desarrollados en lenguajes de más bajo nivel, como FORTRAN y C++, y debido a que ya están implementados y no forman parte del ámbito de esta memoria, en lo restante serán tratados como cajas negras que dadas unas entradas ofrecen unas salidas.

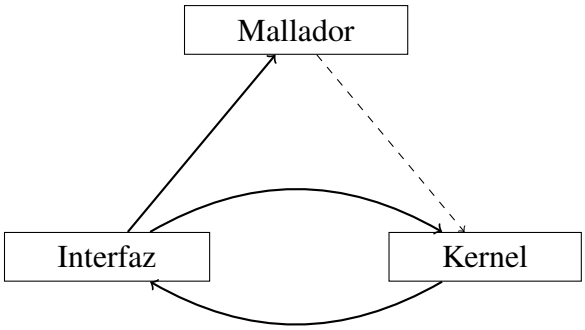


Figura 3.1: Diagrama que representa la interacción entre la interfaz y los kernels

Interfaz de usuario y núcleo se comunicarán por medio de archivos de datos. El kernel recibe unos archivos de entrada, realiza una simulación con esa entrada, y produce una salida. La interfaz de usuario tiene la responsabilidad de generar esos archivos de datos a partir de las opciones que el usuario ha introducido. Una vez el kernel haya terminado su resultado y generado esa salida, la interfaz debe hacer el proceso inverso de interpretar esos resultados. En general cada kernel produce unas salidas de interés para cada tipo de proyecto y el usuario puede acceder a varias pantallas de resultados, cada una centrándose en un tipo de resultados concreto. Más detalle sobre el proceso puede observarse en el esquema de la figura 3.1.

El mallador depende además de otra figura clave que aparece representada igualmente en la figura 3.1, y es el mallador, el cual también debe ser invocado por la interfaz de usuario como paso previo a la simulación. El mallador recibirá las geometrías diseñadas por el usuario e introducidas en el programa y generará con ellas una malla optimizada para realizar los cálculos. El kernel necesita hacer uso de esos archivos de malla para poder simular.

3.2.1. Módulo de guía de ondas en newFASANT 5

newFASANT 5 es una versión anterior del software de newFASANT que se encuentra fuera de soporte ya. newFASANT 5 ofrecía una funcionalidad similar a la que se va a implementar en este trabajo de fin de grado que permitía calcular parámetros en un circuito con alimentación por puertos.

newFASANT 5 fue reemplazado por newFASANT 6, que ha sido reescrito de cero corrigiendo errores e introduciendo mejoras. Uno de los objetivos de este proyecto de fin de grado es ofrecer un reemplazo del antiguo módulo de guía de ondas en la versión actual de newFASANT.

En la versión newFASANT 5, el submódulo de guía de ondas está integrado en el módulo de Análisis de Circuitos. No obstante, desde el punto de vista de la arquitectura interna de código, este módulo estuvo desarrollado completamente independiente al resto de módulos, en particular los proyectos construidos con el submódulo de guías de ondas eran de tipo Waveguide; utilizaba su propia barra de menú y todas las ventanas

de la interfaz de usuario eran propias. Además, se ocupaba de comunicarse con los núcleos y con el mallador.

Cuando el usuario crea un proyecto de guía de ondas en newFASANT 5 dispone de una serie de ventanas y operaciones propias que le permiten agregar dentro de la interfaz guías de ondas como primitivas en un proyecto y luego indicarle al software donde se encuentran los puertos.

Una vez el proyecto contiene las guías de onda y los puertos diseñados se procede a generar los circuitos y a calcular los resultados de la simulación. Cuando estos han sido obtenidos, el usuario puede examinar los resultados. Los resultados que se obtienen en este módulo son:

- El análisis de las matrices de parámetros-S, parámetros-Y y parámetros-Z.
- El análisis de las corrientes a lo largo de las geometrías.
- El análisis de las cargas a lo largo de las geometrías.

En las versiones actuales de newFASANT se obtienen otras propiedades sobre las simulaciones además de las que ya se calculaban en las versiones antiguas, por lo que estos resultados también estarán disponibles para su análisis en el módulo que será desarrollado en este proyecto.

3.2.2. Kernel MONURBS

Como se introdujo anteriormente, los solver son los programas estrella dentro de cualquier herramienta de simulación electromagnética ya que son los encargados de aplicar los algoritmos auténticos para realizar aproximaciones sobre los problemas planteados por los ingenieros.

En el caso del módulo de guías de onda de newFASANT 5, se utiliza un solver integrado en un núcleo denominado MONURBS. MONURBS es una implementación del Método de los Momentos pensada para hacer análisis de estructuras electromagnéticas [16]. Está capacitado para resolver tres tipos de ecuaciones integrales, ya que MoM es un solver integral: EFIE, MFIE y CFIE, ambas basadas en ecuaciones integrales.

- EFIE es la ecuación integral del campo eléctrico, que permite calcular campos eléctricos en función del tiempo, proporcionadas a partir de una corriente J inducida sobre la superficie con la que se está trabajando. EFIE tiene la particularidad de no depender de los límites de la geometría.
- MFIE es la ecuación integral del campo magnético. Está pensada para trabajar con superficies cerradas y obtiene mejores resultados cuando se trabaja con grandes superficies, sobre todo en términos de recursos computacionales, como CPU, empleados [17].

- CFIE es la ecuación integral del campo combinado, que combina de forma lineal las ecuaciones del campo eléctrico con las del campo magnético [18].

MONURBS está pensado para trabajar con geometrías de entrada en formato NURBS, lo que le hace compatible con un rango de aplicaciones de diseño asistido por ordenador que también sean compatibles con NURBS, evitando así trabajar con otro tipo de formatos que puedan provocar problemas. Sólo es necesario que la herramienta CAD utilizada sea capaz de exportar en una especificación de archivo reconocida por el software.

Por otro lado, MONURBS es una herramienta que utiliza distintas técnicas de optimización con el fin de reducir el consumo de recursos que pueda ser necesario para hacerlo funcionar. En particular, MONURBS utiliza paralelización para poder explotar las capacidades de un ordenador multiprocesador por medio de MPI. MPI es una especificación que define una interfaz de paso de mensajes capaz de comunicar procesos [19].

La principal característica de MPI es que evita el uso de memoria compartida cuando se trabaja con sistemas multiproceso. En su lugar, los procesos funcionan independientes entre sí pero utilizan esta interfaz de paso de mensajes para poder mover regiones de memoria de un proceso a otro proceso, mecanismo mediante el cual se transfieren datos entre el proceso maestro y cada uno de los procesos esclavos destinados a efectuar el cálculo.

3.2.3. Trabajo en la interfaz para la alimentación por puertos

La interfaz de usuario de newFASANT 5 es la encargada de ofrecer al usuario un entorno virtual sobre el que construir la simulación que posteriormente será calculada por los distintos núcleos. En consecuencia, uno de los objetivos que debe perseguir es hacer lo más intuitivo posible al usuario la creación de los distintos circuitos que componen el proyecto que va a ser ejecutado.

En el caso de newFASANT 5, el usuario puede agregar un puerto sobre una guía de onda que previamente haya sido construida con la interfaz de usuario o importada desde un archivo externo. El usuario introduce los puertos seleccionando una serie de puntos a través del editor de puertos, como el que se ve en la figura 3.2. Con el editor, el usuario debe seleccionar el límite de la guía de onda sobre la que se pretende introducir el puerto seleccionando una serie de puntos y dejando que la interfaz de usuario calcule el plano de la guía de onda a partir de los puntos seleccionados.

Uno de los inconvenientes de este sistema es su propensión a errores cuando la guía de onda está construida de una forma poco trivial, por ejemplo, teniendo varias aberturas alineadas en el mismo plano, ya que el editor de puertos no siempre es capaz de distinguir los límites de las aberturas simplemente a partir de los puntos seleccionados por el usuario.

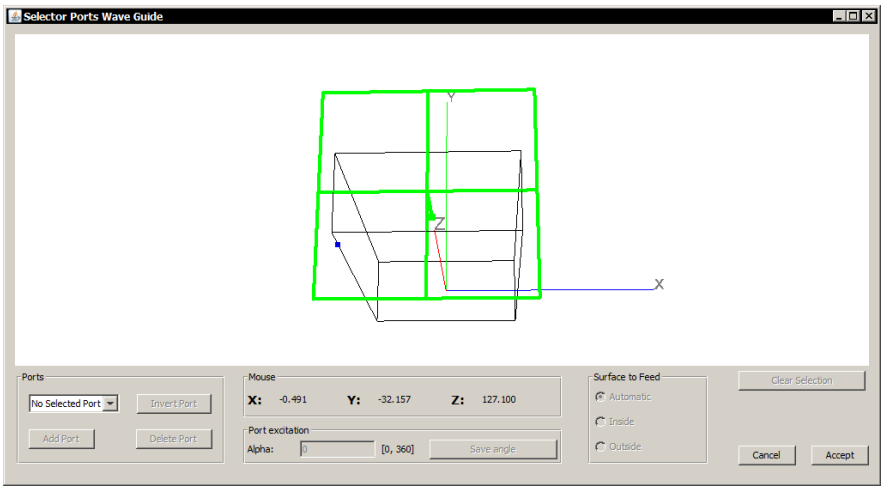


Figura 3.2: Ejemplo de introducción de puertos en newFASANT 5

Este error se considerará de cara a la obtención de la nueva implementación, que utilizará distintos algoritmos geométricos a fin de evitar situaciones como las presentadas anteriormente. En particular, se considerará maximizar el trabajo con NURBS de cara a hacerlo más eficiente y fiable.

3.2.4. Generación de circuitos

Una vez el usuario tiene construido el proyecto que desea simular y tiene introducidos los puertos del proyecto, debe proceder a efectuar los cálculos, invocando antes un paso intermedio que es el de la discretización de las geometrías del proyecto mediante el uso del mallador.

En el caso de proyectos de guía de onda, en newFASANT 5 se aprovecha el paso de mallado para generar y discretizar los distintos circuitos que posteriormente serán ejecutados. Puesto que un proyecto de guías de onda se compone de circuitos, será necesario tratar en algún momento anterior el proyecto con el objetivo de identificar los circuitos que deben generarse para posteriormente ser introducidos en el cálculo. Para identificar los circuitos se hará necesario examinar los puertos incorporados por el usuario.

Un circuito está caracterizado por un puerto principal, que es aquél que en un momento determinado se encuentra excitando, y el resto de puertos secundarios que el usuario haya introducido a través de la interfaz. A cada uno de esos puertos se le asocia un terminal caracterizado como un dipolo, mediante el cual se harán las mediciones de voltajes y corrientes.

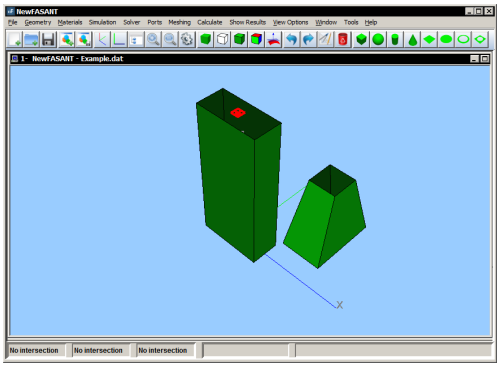
El terminal del puerto principal, además, se comporta como fuente de la alimentación en el solver MONURBS, por lo que es necesario representarlo en la geometría y discretizarlo de igual modo para poder indicarle al kernel que las superficies de dicho puerto son las que están excitadas. En el resto de puertos secundarios no se tiene

que introducir ninguna geometría porque todos los demás terminales se encontrarán inactivos.

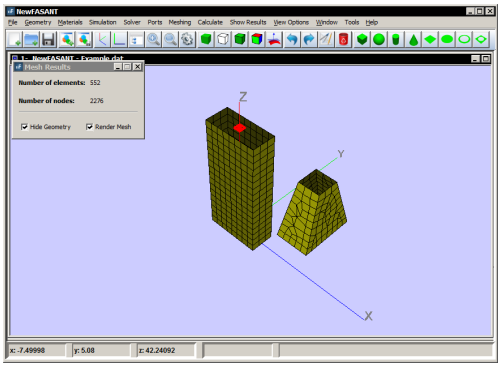
No obstante, debido a la forma en la que funciona el solver de MONURBS, es necesario buscar una manera de cerrar las superficies, y esto se consigue mediante la introducción de tapas que conviertan las guías de onda en cuerpos cerrados. La forma de cerrar las guías no es simplemente introduciendo parches en las aberturas de las guías, ya que entonces las tapas se superpondrían a la posición de los terminales y se producirían problemas en las mediciones.

Es por eso que hace falta prolongar las guías de onda, es decir, hacerlas más largas, para que el circuito final tan en su interior los terminales usados para hacer las mediciones. El procedimiento empleado por newFASANT 5 para hacer las prolongaciones tiene buenos resultados por lo que se usará ese mismo procedimiento en newFASANT 6, generando código que aplique ese algoritmo.

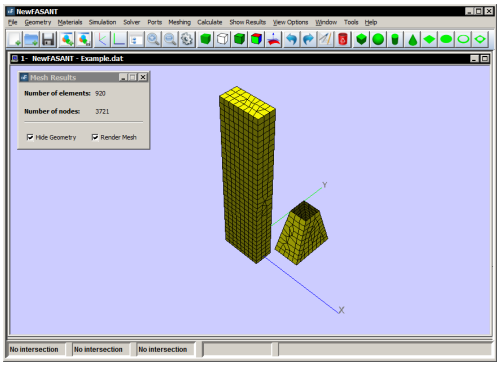
En la figura 3.3 se puede visualizar un ejemplo de generación de circuitos para un proyecto compuesto de dos puertos introducidos en una guía de onda en la subfigura 3.3a. Además de la guía de onda se ha introducido una geometría que no tiene puertos asociados. En la subfigura 3.3b se discretiza el proyecto de forma básica. Las figuras 3.3c y 3.3d muestran como se genera y se discretiza cada uno de los circuitos de los



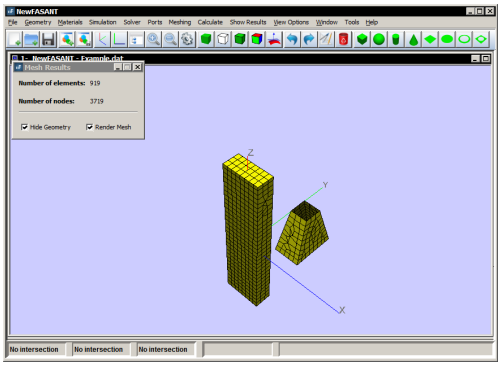
(a) Geometría original tal cual se ha construido con la interfaz de newFASANT 5.



(b) Malla resultado de la discretización de la geometría original.



(c) Malla resultado de discretizar el circuito con el primer puerto como principal.



(d) Malla resultado de discretizar el circuito con el segundo puerto como principal.

Figura 3.3: Circuitos generados a partir de un proyecto base en newFASANT 5.

que se compone el proyecto. En cada circuito, el puerto principal designado cambia y por lo tanto el puerto que recibe la mayor prolongación también cambia.

3.3. Librerías y herramientas utilizadas

3.3.1. Plataforma Java

La interfaz gráfica de newFASANT está desarrollada en el lenguaje de programación Java. Los orígenes de Java se remontan a principios de los años 90 como plataforma orientada a la construcción de aparatos domésticos inteligentes [20] pero pronto encontró su lugar en la Web como herramienta para construir applets y con el tiempo ha ido evolucionando hasta convertirse en una plataforma utilizada en multitud de aplicaciones y dispositivos.

Sin duda, las principales ventajas que motivaron la conversión de la interfaz de usuario de newFASANT a la plataforma Java fueron las siguientes [21]:

- **Es multiplataforma**. El lema de Java tradicionalmente ha sido "escríbelo una vez, ejecútalo en cualquier parte". Java es una plataforma interpretada. El programador escribe su código y genera un código máquina independiente de la plataforma que es procesado por una máquina virtual de Java, que intermedia con el hardware real. Esto permite que haya implementaciones de la máquina virtual de Java para muchas plataformas, incluyendo los principales sistemas operativos del mercado.
- **Está orientada a objetos**. La orientación a objetos es uno de los paradigmas de programación más empleados en la industria del software. Java utiliza la orientación a objetos de forma nativa incluyendo buenas prácticas aceptadas en el desarrollo de software. Esto permite además integrarlo con otras técnicas de desarrollo de software como el UML.
- **Tiene una gran librería estandar**. Java dispone de una librería estandar para facilitar el trabajo con tipos de datos complejos. Además, Java utiliza su propio sistema de representación de interfaces gráficas, Swing, mediante el cual es posible construir ricas interfaces de usuario que se pueden visualizar en cualquier plataforma compatible.
- **Soporte para librerías de terceros**. Es fácil utilizar librerías de terceros para agregar funcionalidad que no ofrezca la librería estandar de Java y que no quiera ser desarrollada. newFASANT hace uso de algunas de estas librerías para evitar reinventar la rueda.
- **Es abierto**. La especificación de la máquina virtual de Java es accesible de forma gratuita [22], y es posible construir implementaciones de esta máquina virtual. Desde su versión Java 7, Java SE está basada en OpenJDK [23], una implementación de Java de código abierto desarrollada inicialmente por Sun.

3.3.2. Java3D

Java3D es una librería desarrollada inicialmente por Sun y actualmente mantenida por Oracle que permite trabajar con representaciones 3D de mundos virtuales [24]. Java3D se apoya en las otras librerías gráficas de la plataforma Java, AWT y Swing, usadas para representar interfaces gráficas, y proporciona un mecanismo sencillo para incluir las representaciones de esos escenarios en ventanas gráficas [25].

Java3D se apoya en DirectX y OpenGL, tecnologías que permiten ofrecer características gráficas avanzadas en los sistemas operativos. DirectX estará disponible en Windows, mientras que OpenGL será usado para representar los escenarios de Java3D en los sistemas operativos Linux y MacOS X.

La API que ofrece Java3D está basada en un grafo de escena, es decir, una representación de los elementos del mundo virtual como un grafo dirigido donde existen distintos **nodos** con unas propiedades concretas [26]. Algunos de los nodos representan **primitivas**, como esferas, cajas o triángulos, que pueden ser manipuladas por el programador definiendo otros nodos especiales, tales como **materiales** y **apariencias**.

También destaca el uso de grupos para asociar algunos de estos nodos, componiendo un grafo final en forma de árbol donde el nodo raíz es el propio escenario virtual, compuesto de hijos, que son grupos de nodos.

3.3.3. MPICH2

MPICH2 es una implementación de la especificación MPI. Por medio de MPICH2 los desarrolladores tienen a su disposición una forma de aplicar las técnicas descritas por dicha especificación de modo que distintos procesos que no compartan memoria de por sí puedan intercambiar datos por medio de un paso de mensajes.

newFASANT utiliza MPICH2 para paralelizar los kernels y el mallador. El usuario especifica antes de iniciar una operación intensiva con la CPU cuántos procesos desea utilizar para paralelizar la operación, y se inician de forma paralelizada los procesos que haya especificado el usuario. Esto permite que en un superordenador compuesto de varios procesadores, se puedan aprovechar todos. El rendimiento será el adecuado cuando se seleccionen tantos procesos paralelos como núcleos tenga el ordenador en el que se esté ejecutando la simulación. Indicar menos procesos que núcleos hará que parte de los recursos de la máquina no se destinen a efectuar el cálculo con la mayor rapidez posible; pero indicar más procesos será contraproducente en tanto que cada CPU tendrá que intercalar la ejecución de varios procesos destinados a hacer cálculos, lo que hará perder tiempo debido al intercambio de contextos al ejecutar el proceso.

4 Desarrollo experimental

El objetivo de esta sección es detallar el proceso de diseño e implementación de la funcionalidad que va a ser incorporada en newFASANT 6 para el trabajo con guías de onda y alimentación por puertos.

4.1. Modelo de desarrollo

El ciclo de desarrollo que se va a emplear es el iterativo. Este ciclo de desarrollo presenta más ventajas que un ciclo de desarrollo tradicional, como el de cascada, como se estudiará a continuación. La principal ventaja de cara al desarrollo de esta funcionalidad completa es la posibilidad de introducir cambios en el diseño del sistema, y en poder realizar una vista general del proyecto a medida que se va desarrollando.

El ciclo de **desarrollo en cascada** es el tradicionalmente empleado por la industria del software durante la última mitad de siglo. En un ciclo de desarrollo en cascada se definen varias etapas en el desarrollo de software: análisis de requisitos, diseño, implementación, pruebas y mantenimiento. No obstante, este ciclo de vida tiene dos grandes inconvenientes. El primero, que no puede comenzar una etapa hasta que no termine la anterior, lo que alarga el ciclo de desarrollo. Y segundo, que resulta raro que los requisitos de un software se mantengan estáticos. Bien sea por olvidos, o bien sea porque el cliente así lo pide, lo normal es que se tengan que introducir cambios en un sistema de software incluso durante el desarrollo. El ciclo de software en cascada es poco flexible con estos cambios.

En el ciclo de **desarrollo de software iterativo** o incremental, el desarrollo se divide en iteraciones, con una duración aproximada de entre un par de semanas a un par de meses. Y en cada una de estas iteraciones, se define un ciclo de vida en cascada como el presentado anteriormente pero a nivel local.

De este modo, a partir de una lista de requisitos que debe cumplir el sistema final, se puede determinar al inicio de cada iteración cuáles son los ítems críticos en los que una iteración se va a centrar, y se va a perfilar esa parte del sistema. Al final de la iteración, se realizará un estudio de la iteración en busca de posibles cambios que haya que introducir en el sistema, bien detectados por el equipo de desarrollo (como un mal diseño) o bien proporcionados por el cliente (como un cambio de última hora que éste solicite). Estos cambios afectarán a la siguiente iteración, de modo que a la hora de analizar los ítems críticos de la siguiente iteración esos cambios serán considerados.

4.2. Pruebas de software

4.2.1. Introducción a las pruebas

Las pruebas de software son una métrica utilizada para mejorar la calidad del software [27]. La intención de las pruebas de software es facilitar la detección de **defectos** que ayuden a subsanarlos para verificar el correcto funcionamiento de un programa de software.

Las pruebas de software pertenecen a una de las etapas del ciclo de vida del software. Los principales ciclos de desarrollo empleados en la industria del software incluyen en algún momento del desarrollo las pruebas para verificar el comportamiento del software y asegurar su correcto funcionamiento.

En el caso del ciclo de vida en cascada, por ejemplo, la verificación es una de las últimas etapas del desarrollo de software. O, en el caso del ciclo de software iterativo, al final de cada iteración también se presenta una fase de verificación donde se buscan defectos que puedan ser localizados para la siguiente iteración.

Algunas metodologías llevan las pruebas más allá haciendo que formen parte de forma más proactiva del desarrollo de software. Es el caso, por ejemplo, del **Desarrollo orientado a pruebas** propio de metodologías de **Programación Extrema**.

En cualquiera de los casos, las pruebas de software tienen distintas funciones, asociada cada función con un tipo distinto de prueba de software. La misión de las pruebas de software es [28]:

- Verificar la correctitud de los distintos componentes que forman un sistema de software.
- Verificar la apropiada integración de esos componentes dentro del software.
- Verificar que el software se adecúa perfectamente al entorno sobre el que va a estar operando.
- Verificar que el software cumple los requisitos y las expectativas del cliente.
- Verificar que cualquier cambio hecho sobre un componente no afecta a ninguno de los componentes ya desarrollados y probados con anterioridad.

La métodología orientada a pruebas indicada anteriormente utiliza un tipo de desarrollo conocido como **luz roja, luz verde**. El objetivo de la metodología orientada a pruebas es desarrollar primero una prueba unitaria y después la implementación de la clase que hace funcionar esa prueba unitaria. De este modo, el desarrollador puede utilizar la prueba unitaria para definir una especificación de los comportamientos que debe tener su clase y utilizarla para verificar que la clase lleva a cabo correctamente esa especificación.

El modelo **luz roja, luz verde** se compone de dos etapas:

- En la fase de luz roja, el programador implementa la prueba unitaria en primer lugar, construyendo una prueba que verifica una funcionalidad que no ha sido implementada, y la intenta probar. Por trivial que parezca es necesario comprobar que el resultado de la prueba sea un error, representado como una luz roja. Los motivos que hacen que la prueba falle son diversos.

  - La clase o el método que se está intentando probar aún no existe. En consecuencia, la prueba unitaria no es capaz de funcionar ya que el compilador de Java no podrá localizar la clase o el método indicado.
  - El método que se está probando ha sido creado pero su comportamiento aún no ha sido definido, de modo que no hace lo que tiene que hacer y por lo tanto la prueba falla porque el resultado esperado no se ha producido.

- En la fase de luz verde, el programador implementa el código que hace que la prueba unitaria funcione. Por ejemplo, crea la clase o el método que aún no existe o implementa el algoritmo que hace que el método devuelva lo que debe devolver.

Este proceso se repite de forma iterativa. De modo que una vez se ha cumplido una prueba unitaria, se pasa a implementar la siguiente, que debe volver a verificar el mismo proceso. En ocasiones incluso puede ser necesario que para un mismo método se hagan varios ciclos de luz roja, luz verde.

El modelo de desarrollo orientado a pruebas tiene como ventajas:

- La introducción del programador en la cultura de las pruebas como elemento que permite detectar la existencia de fallos en una aplicación de software. Si el programador cuenta con una especificación que le permite conocer qué debe hacer la clase que debe implementar, resulta fácil verificarla sobre la marcha.

- La elaboración de implementaciones concisas y reducidas. Debido a que la prueba unitaria contiene una especificación de lo que debe hacer la clase, el programador escribirá únicamente los métodos que permitan que las pruebas unitarias validen. El programador no tiene permitido agregar código que no forme parte de una prueba unitaria y eso evita construir métodos que no formen parte de la especificación original. Por ejemplo, métodos que el programador haya implementado por conveniencia pero que no sean realmente utilizados.

4.2.2. Estado de pruebas en newFASANT

En este momento la base de código de newFASANT no tiene automatizadas sus pruebas de software. En el desarrollo de la aplicación las pruebas se realizan de forma

manual mediante la ejecución de casos, la comprobación en la correctitud de los resultados obtenidos y pruebas de estrés aplicadas a proyectos complejos para asegurarse de que no producen problemas con la interfaz de usuario.

Los criterios de desarrollo del módulo presentado en esta memoria consideran algo natural el desarrollo de pruebas de software y por lo tanto se introducirán sobre el código nuevo que se desarrolle. En la sección 4.2.3 se detalla el plan de pruebas que se va a emplear.

Agregar pruebas unitarias es una inversión de recursos que beneficia cuando sirve para detectar errores que de otro modo hubiesen tenido que detectarse de forma manual, ya que de no tenerlas a menudo se obliga a invertir más recursos y más tiempo para localizar el error y subsanarlo.

Sin embargo, no se pueden agregar pruebas unitarias indiscriminadamente sobre una base de código que no ha sido probada. En general sólo se puede introducir en momentos de cambio: cuando se agrega una característica al software, o cuando se busca modificar el software, por ejemplo para corregir un error o para mejorar un diseño. Es en ese momento cuando, siempre sobre la base de código concreta que va a cambiar, se puede considerar la inclusión de pruebas unitarias. El proceso es el siguiente:

1. Si es necesario, introducir los cambios mínimos que permitan agregar la primera prueba unitaria.

2. Diseñar la prueba unitaria sobre el código que se quiere cambiar.

3. Realizar el cambio.

4. Verificar que la prueba unitaria pasa sobre el código cambiado.

Para realizar el cambio, se realizan **refactorizaciones**, que son mecanismos que permiten cambiar de una forma predecible una base de código asegurando que los cambios internos no afectan a la forma con la que otras clases interactúan con el software. A día de hoy los entornos integrados de desarrollo proporcionan potentes mecanismos automatizados de refactorización que permiten hacer modificaciones sobre código y corregir cualquier dependencia en otras clases que deba ser cambiada también.

Se espera que dado que no hay antecedentes recientes de uso de pruebas de software automatizadas dentro de la base de código, esta acción sirva de precedente y anime posteriores desarrollos orientados a pruebas con el fin de mejorar la calidad del software.

4.2.3. Plan de pruebas

En el desarrollo de este módulo, se trabajarán con tres tipos de pruebas:

- **Pruebas unitarias**: el código desarrollado utilizará pruebas unitarias. La lógica de negocio se desarrolla utilizando una metodología orientada a pruebas y por lo tanto antes de agregar la implementación se diseñará la prueba.
- **Pruebas de integración**: las pruebas de integración serán utilizadas cuando se quiera verificar que el módulo se ensambla correctamente con el resto de componentes; principalmente con el mallador, el núcleo y el sistema de archivos cuando tiene que escribir archivos.
- **Pruebas de aceptación**: el módulo se presenta ante el usuario final como un conjunto de paneles que utiliza para interactuar con las clases. Por lo tanto, será necesario realizar pruebas asegurando el correcto funcionamiento de la interfaz gráfica desarrollada.

4.3. Desarrollo del paquete de primitivas

Las primitivas son las geometrías que el usuario puede incorporar en la simulación. Si bien el usuario tiene a su disposición los comandos que le permiten construir con precisión una guía de onda de forma manual, la misión de las primitivas es automatizar la construcción de las principales guías de onda.

El diseño del paquete de primitivas se divide en dos partes: primero, las propias geometrías, incluyendo sus estructuras de datos y su representación de Java3D; y después, los comandos que permiten construir esas geometrías a través de la consola de la interfaz de usuario.

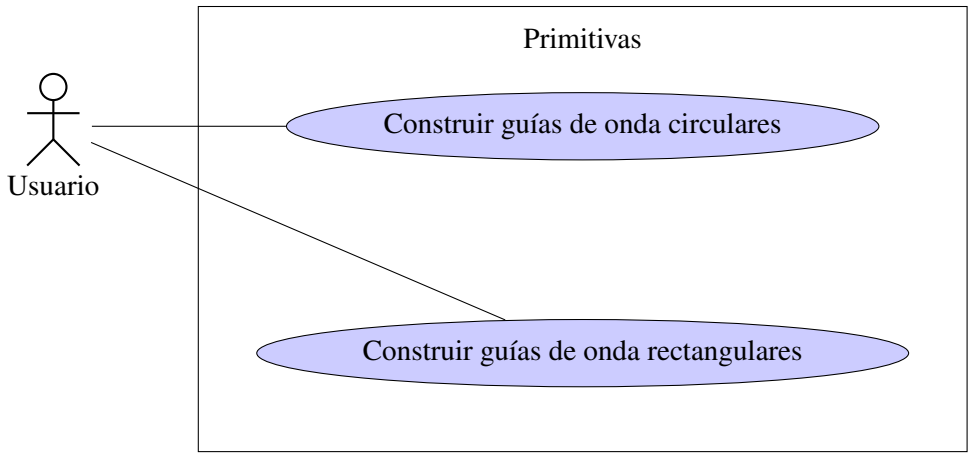


Figura 4.1: Diagrama de casos de uso para el paquete de primitivas.

En la figura 4.1 se presenta el diagrama de casos de uso correspondiente a esta sección.

Se dará soporte a dos tipos de guía de onda: rectangulares y circulares. Sin embargo, el paquete se diseña de forma extensible para permitir construir más guías de onda si se diese la necesidad.

4.3.1. Diseño de las primitivas

Diseño general de la geometría en newFASANT 6

newFASANT utiliza Java 3D para representar las geometrías en la interfaz de usuario. En Java 3D, el usuario define un universo virtual (`SimpleUniverse`) sobre el que se agregan los objetos a representar como **ramas de contenido**. Java 3D es una librería que define lo que se representa por pantalla usando un grafo de escena, es decir, los distintos elementos que definen el universo virtual se gestionan como un grafo donde unos nodos hoja representan la unidad más básica de primitiva que puede representarse en pantalla (por ejemplo, un cubo, o un array de puntos), y estas hojas se agrupan para construir escenarios más complejos.

En el caso del universo virtual representado por la interfaz gráfica de newFASANT, el diseño de la aplicación considera los objetos geométricos a representar por pantalla (como superficies u objetos compuestos) como subtipos de `BranchGroup`, una de las clases de la librería Java 3D que se comporta como un grupo, manteniendo varios nodos hijo dentro del grafo.

Eso significa que todo lo que para newFASANT sea representable, será de tipo BranchGroup, por lo que podrá ser agregado a la interfaz de usuario mediante la librería Java3D. No obstante, la librería estandar de newFASANT ya dispone de su propio sistema de geometrías por encima de Java3D.

Objeto3D es la clase de la que deben extender las geometrías compuestas, incluyendo las guías de onda construidas en este paquete. Puesto que los objetos 3D son especializaciones de la clase `BranchGroup`, disponen de los métodos más importantes que le permitirán a la clase definir su geometría Java 3D:

`addChild(n : Node)`
: Con este método un `Group` puede agregar un nodo hijo dentro de su rama del grafo.

`removeChild(n : Node)`
: Permite eliminar un nodo concreto de la lista de nodos.

`removeAllChildren()`
: Permite eliminar todos los nodos hijo de la lista de nodos.

Las clases que se comporten como geometrías son responsables de definir su apariencia incluyendo un método que construya los nodos Java 3D hijos y que los agregue a su árbol de nodos. Afortunadamente la librería de newFASANT ya dispone de métodos en las clases que gestionan curvas y superficies NURBS para representar una curva o una superficie como un nodo de Java 3D por lo que en la mayoría de los casos bastará con obtener las superficies o curvas NURBS y extraer la representación Java 3D ya compilada.

Diseño general de la consola de newFASANT 6

Será necesario definir una serie de comandos para permitir instanciar guías de onda. La consola de newFASANT 6 dispone de una API para definir comandos que pueden ser habilitados al inicio del programa o sólo cuando un determinado módulo se cargue.

La clase de la que todo comando tiene que extender es `Command`. Esta clase es abstracta y dispone de una serie de métodos a implementar. La consola los invocará en el momento oportuno.

`execute(ConsolePanel, List<String>)`
: Este método es invocado cuando el usuario teclea un comando y pulsa ENTER. El usuario puede introducir argumentos en la escritura del comando por lo que se le pasan al método para que los pueda procesar.

`processCommand(ConsolePanel)`
: Este método ejecuta la funcionalidad principal del comando y es invocado una vez todos los parámetros del método están procesados y se conoce con exactitud todo lo que necesita el comando para funcionar (como por ejemplo una superficie o una curva que tenga que manipular).

`updateCommand(ConsolePanel)`
: Este método es invocado cuando haya que actualizar un comando. La actualización tiene lugar cuando el usuario modifique los parámetros con los que ha sido invocado un comando por medio del historial de la aplicación. En este método el comando debe repetir su operación utilizando los nuevos parámetros.

`getResulObjects() : ArrayList`
: Este método devuelve el conjunto de geometrías generadas por el comando si las hay. De este modo la interfaz de usuario de newFASANT puede incorporar las geometrías generadas al proyecto. Los comandos que generen guías de onda devolverán aquí la guía de onda generada.

`help() : String`
: Devuelve el mensaje de ayuda que se mostrará al usuario cuando invoque el comando en modo de ayuda. Se incluirá una descripción de lo que hace el comando y los posibles parámetros que tenga.

`getScriptCommand() : String`
: Devuelve una representación del comando que sea compatible con el sistema de scripts. La consola puede importar scripts que sean ejecutados de forma secuencial. Cuando el usuario exporta un script a partir de un proyecto, este método devuelve el comando que permitirá volver a ejecutar el comando al importarlo.

`undo(ConsolePanel)`
: Este evento es ejecutado por la interfaz de usuario cuando el usuario presiona el botón de Deshacer. Debe reponer la situación actual previa a la ejecución del comando (por ejemplo, devolver un objeto eliminado, eliminar un objeto introducido en la interfaz...).

`redo(ConsolePanel)`
: De forma análoga, este evento es ejecutado por la interfaz de usuario al pulsar el botón de Rehacer. Debe repetir la funcionalidad que previamente fue deshecha, por ejemplo, eliminando de nuevo un objeto eliminado o volviendo a incorporar el objeto a la interfaz.

Diagrama de clases

En la figura 4.2 se presenta el diagrama de clases del paquete de primitivas.

Las guías de onda se agrupan en torno a la clase abstracta `GeoWaveguide`, que representa una guía de onda de cualquier tipo de forma abstracta. Su propiedad común es el **centro** de la guía de onda, que determina en qué posición del escenario virtual se representa.

Por medio del getter `getCenter` y del setter `setCenter` se puede obtener y modificar el centro de la geometría. En general, cualquier operación que modifique la geometría de una guía de onda deberá asegurar la correcta actualización de la representación 3D de la geometría para actualizar lo que se ve a través de la interfaz. En consecuencia:

- El mutador `setCenter`, y en general cualquier mutador que utilicen las clases que hereden de `GeoWaveguide` deberán actualizar la apariencia 3D de la guía de onda cuando la modificación se hace efectiva.
- El accesor `getCenter` nunca devuelve la referencia a su propia instancia de centro, ya que eso podría provocar que código cliente usase esa referencia para modificar el centro de la geometría sin actualizar la apariencia 3D. En su lugar, si el accesor no devuelve tipos primitivos sino referencias, se debe trabajar siempre con copias, o siempre debe requerir objetos como parámetro que sean modificados por el propio mutador. Es el caso de `getCenter`, que recibe como parámetro un punto cuyas componentes serán modificadas para que coincidan con el centro real de la geometría.

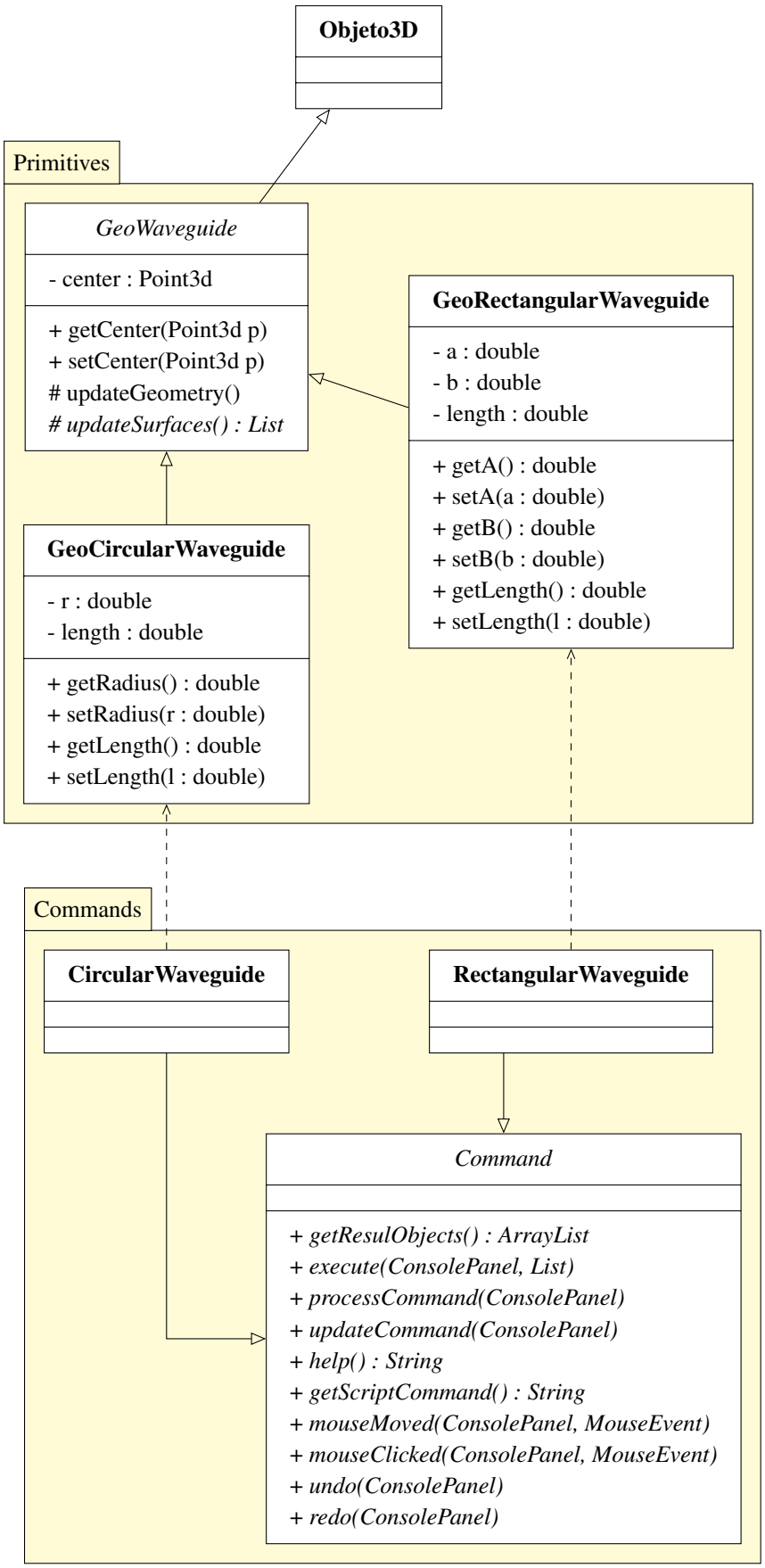


Figura 4.2: Diagrama de clases del paquete de primitivas

Los métodos abstractos que expone esta clase y que han de ser implementados por las guías de onda son:

`updateSurfaces() : List`
: Este método deberá generar la superficies NURBS que definen a la guía de onda construida y devolverlas en una lista. De este modo, cada clase es responsable de definir su propia representación parametrizada.

Finalmente, el método `updateJava3DGeometry()` permite reconstruir la geometría de Java3D que luego es representada en la interfaz de usuario. Invoca el método `updateSurfaces` y luego genera con las superficies NURBS recuperada los distintos nodos de Java 3D que representen las superficies, agrupándolas en el `BranchGroup` asociado al objeto 3D.

4.3.2. Implementación de las primitivas

Los primeros componentes implementados en este proyecto serán las primitivas, ya que no tienen dependencias de otras partes de este proyecto y son lo suficientemente rápidas de desarrollar como para completar en una iteración mientras las siguientes etapas del proyecto se diseñan.

Tal como se indicó en secciones anteriores, las guías de onda son un componente electromagnético más que se integra junto al resto de los que ya dispone este programa (como por ejemplo otro tipo de antenas), por lo que dentro del MOM, compartirán la lógica de negocio de algunas cosas que hagan ya otro tipo de antenas (como por ejemplo el mallado).

En la base teórica que se presentó al principio de este trabajo de fin de grado se enumeraron los distintos tipos de guía de onda presentes en el mercado en mayor proporción, y se indicaron las diferencias entre cada tipo, la forma de cada sección y cualquier otra particularidad. Ahora, nos centraremos en cómo se puede representar por medio de objetos NURBS esas guías de onda.

Guías de onda rectangulares

El primer tipo de guía de onda que se va a implementar es la guía de onda rectangular. De acuerdo con lo que se acordó en fase de diseño, el objeto `GeoRectangularWaveguide` será un objeto de tipo `GeoWaveguide`.

En primer lugar implementaremos el método que genera las NURBS de esta guía de onda. Para ello, vamos a considerar siempre como referencia una guía de onda que esté orientada longitudinalmente al eje Z, es decir, le damos al usuario la tarea de rotarla para que esté en la posición que él espera que esté.

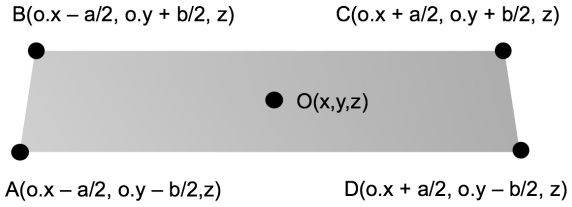


Figura 4.3: Obtención de los puntos que delimitan la base de una guía de onda rectangular

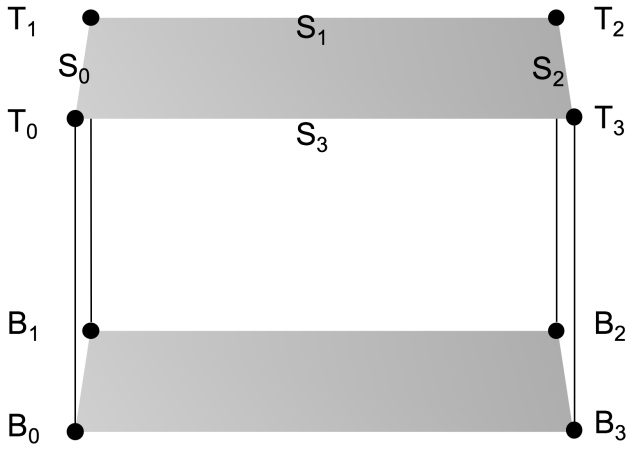


Figura 4.4: Obtención de los puntos que delimitan la base de una guía de onda rectangular

El proceso se divide en tareas simples. Primero se buscará una forma de obtener, dada una altura Z, los cuatro puntos que definen los bordes de una guía de onda a esa altura. Hecho eso, se generarán los puntos del borde inferior y del borde superior y se construirán las facetas que conectan los puntos de ambos extremos, como si de una operación de extrusión se tratase.

En la figura 4.3 se presenta el proceso por el que se obtienen los cuatro puntos a partir del centro. A partir de una función que devuelva esos cuatro puntos para una altura Z, podemos obtener la que permite combinar los puntos de la base inferior con los de la base superior y construir las facetas. El proceso de construcción de facetas debe ser preciso para seleccionar los puntos en el orden correcto y para que las normales de las superficies se generen en la dirección correcta.

En la figura 4.4 se visualizan los puntos de ambos bordes y las conexiones que deben establecerse. El proceso es más sencillo de lo que parece, ya que si generamos la base inferior a la altura indicada por la componente Z del centro, y la base superior desplazada sobre la componente Z del centro el valor que indique la altura de la guía de onda, entonces se aprecia que cada una de las cuatro aristas que compondrán la guía de onda estará determinada únicamente por el par de puntos inferior y superior que

tengan el mismo índice del array. Por ejemplo, sea `b[]` los puntos de la base inferior y `t[]` los de la base superior. En ese caso las aristas se definirán por los pares de puntos `(b[0], t[0])`, `(b[1], t[1])`, `(b[2], t[2])` y `(b[3], t[3])`.

Para generar las superficies usamos una función de la librería NURBS de newFASANT que permite construir facetas. Esta función es `createFacet(p1, p2, p3, p4)`, a la que se le pasan los cuatro puntos y que generan la superficie NURBS correspondiente. Por lo tanto, para definir las cuatro superficies que componen una guía de onda rectangular se construye una lista con las superficies generadas por la llamada a esta función cuando le pasamos la tetrada de puntos que componen cada superficie.

Por último, quedará por describir la implementación de los distintos métodos accesores (get) y mutadores (set) para las propiedades de esta guía de onda rectangular. La implementación no aprecia particularidades especiales y es directa, pero se va a hacer mención a la forma de verificar que los parámetros sean válidos.

Cada parámetro tendrá una serie de restricciones, como por ejemplo que no se le indiquen valores negativos. Los mutadores examinan que las precondiciones se cumplan y en caso de que no lo hagan generarán una excepción de tipo `InvalidArgumentException` para indicar que el parámetro es incorrecto.

Guías de onda circulares

La implementación de la construcción de las superficies NURBS en guías de onda circulares será distinta. En este caso, se generará una circunferencia utilizando curvas NURBS. Esta se hará mediante la generación de cuatro arcos.

La librería NURBS de newFASANT ofrece una función denominada `createArc (center, radius, thi, thf)` que permite generar arcos situados en torno a un centro del arco y con un radio determinado. Por medio de los parámetros `thi` y `thf` se indica el ángulo inicial y el ángulo final del arco.

A partir de estos parámetros, la librería de newFASANT determinará los dos puntos que delimitan el arco a partir del centro, el radio y cada uno de los ángulos, generando los puntos inicial y final. Con los dos puntos determinados, termina de construir el arco.

En el caso de la guía de onda circular, se dividirá la sección de la geometría en cuatro arcos, con una distancia angular de $\pi/2$. Cada uno estará orientado en un cuadrante distinto del plano XY, de modo que los cuatro arcos tendrán los siguientes parámetros de ángulo inicial y final.

- $\theta_i = 0$, $\theta_f = \frac{\pi}{2}$
- $\theta_i = \frac{\pi}{2}$, $\theta_f = \pi$

- $\theta_i = \pi$, $\theta_f = \frac{3\pi}{2}$
- $\theta_i = \frac{3\pi}{2}$, $\theta_f = 2\pi$

Una vez se han construido las curvas, se las aplica una operación de extrusión para obtener las superficies NURBS. El vector desplazamiento aplicado a la extrusión será el vector unitario del eje Z multiplicado escalarmente por la longitud de la guía de onda, de modo que las superficies generadas tendrán una altura L y una vez agrupadas formarán un cilindro.

Comandos para trabajar con guías de onda

Se busca construir unos comandos que permitan al usuario construir guías de onda rectangulares y circulares utilizando la consola. Para ello, el usuario tendrá que introducir como parámetros, en el caso de las guías de onda rectangulares, el centro, las dimensiones de A y de B deseadas, y la altura de la guía de onda. En el caso de guías circulares se introducirá en su lugar el centro, el radio de la guía y también la altura.

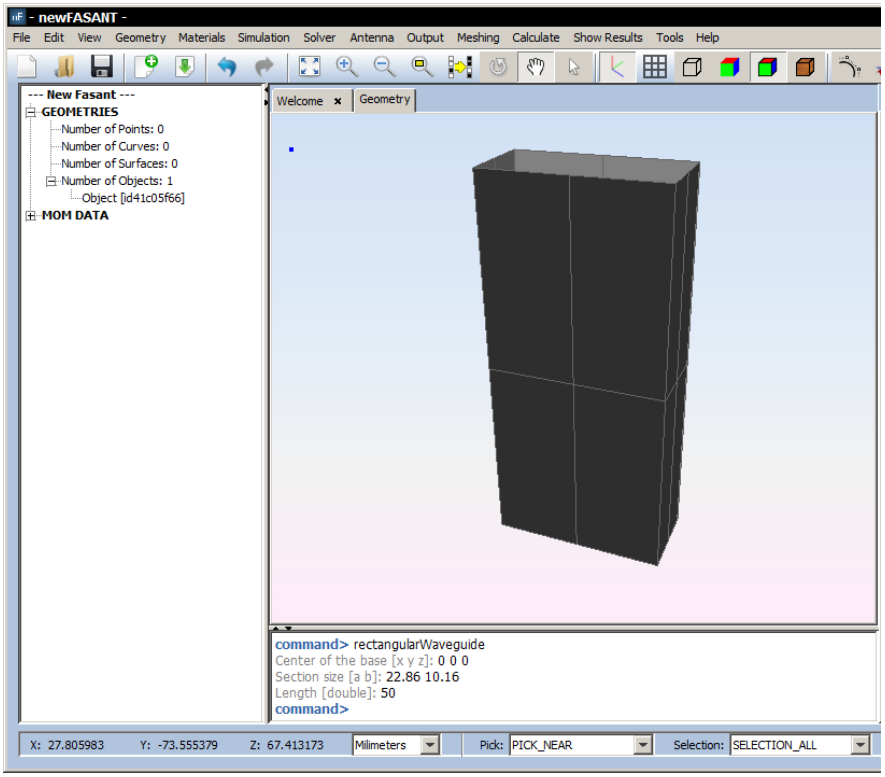
Para ello se construyen dos clases de tipo comando, que extiendan a `Command`. El comportamiento de ambas clases es parecido por lo que se presentarán ambos comandos a la vez.

En ambos casos es necesario identificar las propiedades de las que depende la guía de onda e introducirla en el mapa de parámetros del comando. De este modo, la interfaz de usuario y el propio comando podrán manipular los parámetros y regenerar los objetos cuando lo necesiten. El comando puede acceder a este mapa de parámetros para obtener los valores durante la construcción.

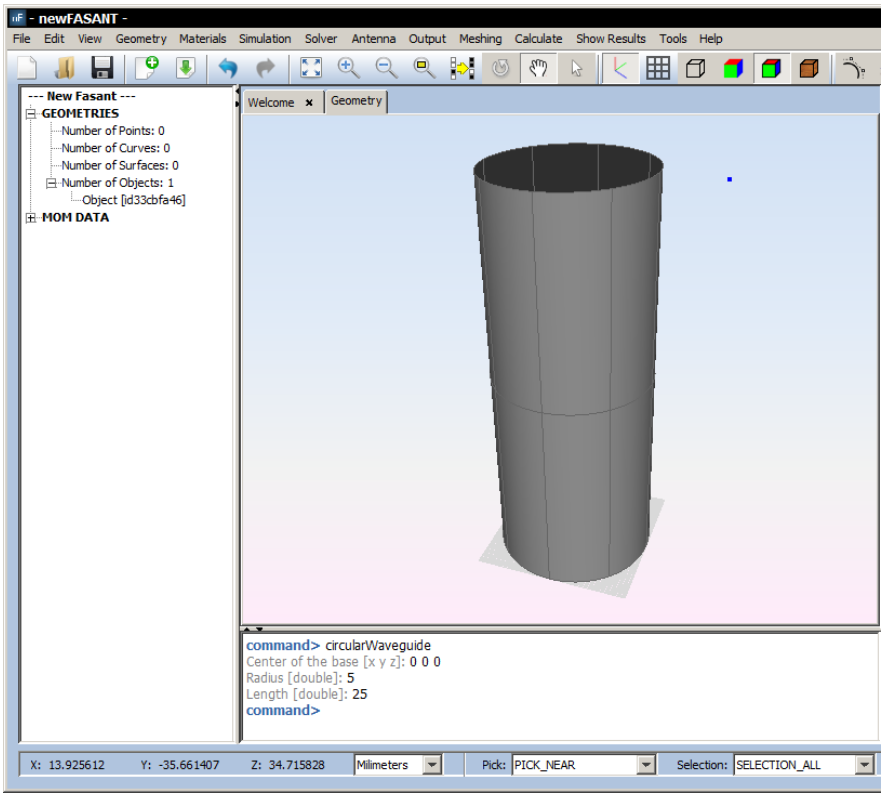
Finalmente, cuando se invoca el código principal del comando y se genera la geometría, se almacena en uno de los campos del comando a fin de no perder su referencia, especialmente cuando se llamen a métodos como `undo`, `redo` o `getResulObjects`. Una vez en newFASANT se ejecuta un comando, se construye una instancia de ese mismo comando, por lo que siempre es posible para la interfaz de usuario ejecutar cualquiera de los métodos (como el undo o redo) accediendo a un comando almacenado en el historial.

Por último, se hace esos comandos accesibles cuando el usuario inicia un proyecto de MOM y se eliminan esos comandos cuando el usuario cierra el proyecto de MOM.

El resultado final es que si el usuario crea un proyecto MOM puede utilizar los comandos `rectangularWaveguide` y `circularWaveguide` para construir geometrías, como se puede ver en los ejemplos presentados en la figura 4.5.



(a) Guía rectangular



(b) Guía circular.

Figura 4.5: Comandos para trabajar con guías rectangulares y circulares.

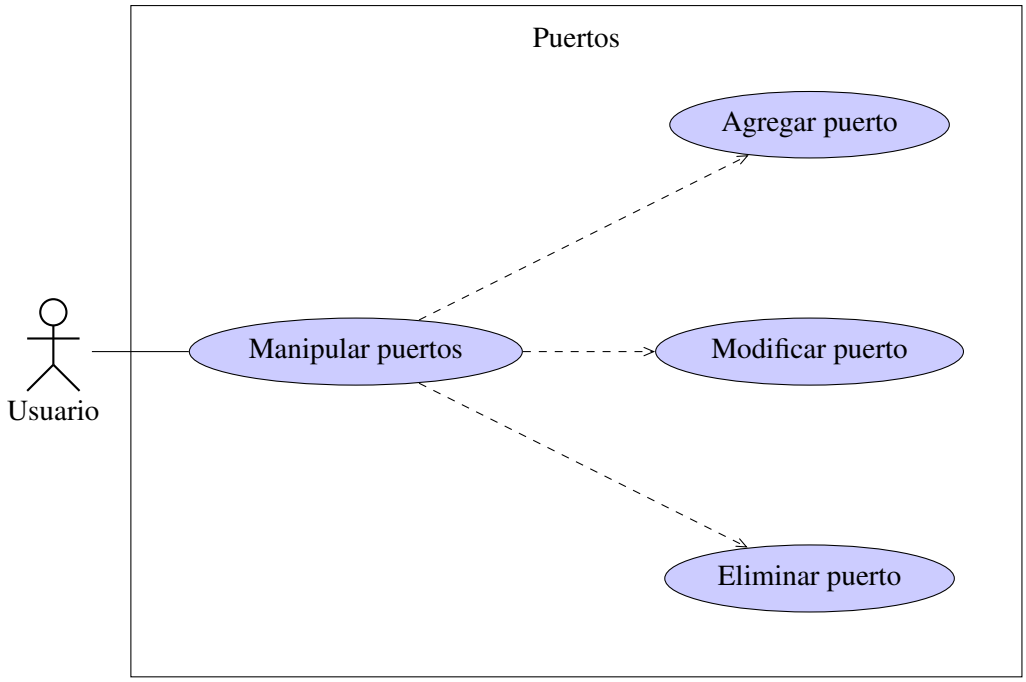


Figura 4.6: Diagrama de casos de uso del paquete de puertos.

4.4. Diseño del paquete de puertos

El objetivo del subsistema de puertos es permitir al usuario agregar puertos a las guías de onda que previamente ha creado o importado. Los puertos contienen los dipolos que hacen falta para poder realizar la simulación por lo que es obligatorio que el usuario los agregue para poder proceder con el mallado.

El usuario debe ser capaz de:

- Agregar puertos sobre una geometría constituida como guía de onda.
- Modificar los ajustes de los puertos definidos sobre la geometría.
- Eliminar los puertos agregados previamente a una guía de onda.

Sobre lo que se considera geometría de guía de onda, sería un error considerar únicamente las primitivas construidas con los comandos previamente diseñados como guías de onda. El usuario debería poder construir a mano la guía de onda también. Por ejemplo, el usuario debería poder aplicar la operación `rectangle` seguida de la operación `extrude` para definir la guía de onda. Además, también debe tener libertad para importar geometrías en otros formatos de intercambio que también representen guías de onda.

Cuando el usuario quiera incorporar un puerto de guía de onda sobre un objeto ya construido, lo hará seleccionando la zona de la geometría en la que lo vaya a introducir. Los puertos **se incorporan sobre los extremos de una guía de onda**. Es decir, cuando

el usuario agrega los puertos, lo hace sobre sobre la sección de uno de los extremos de la guía de onda. El objetivo de esto es poder agregar varios puertos y luego estudiar la forma en la que se acoplan de dos en dos.

4.4.1. Diseño de la alimentación por puertos

A pesar de que en la sección anterior se han incorporado una serie de primitivas con las que el usuario puede construir geometrías en la interfaz de usuario, no se debe olvidar que el usuario tiene libertad para importar los modelos de las guías de onda desde un programa CAD o para diseñarlas manualmente usando otros comandos de la interfaz de usuario de newFASANT, como un rectangulo al que se le aplica una operación *extrude*.

En definitiva, pese a que se incorporan al programa geometrías de guía de onda para facilitar su construcción, no se puede limitar a este tipo de geometrías, lo cual exige a trabajar únicamente con propiedades NURBS en vez de aprovechar las propiedades propias de las guías de onda como su ancho, su altura...

En el módulo de guías de onda desarrollado para newFASANT 6, se aprovechará al máximo las capacidades geométricas NURBS permitiendo al usuario trabajar de forma nativa con las propias curvas de contorno de la geometría. El usuario seleccionará las curvas que conforman el borde de la guía de onda, formando un lazo cerrado. Con esto el usuario puede definir de una forma intuitiva **dónde** quiere agregar el puerto ya que tiene que seleccionar la abertura por la que se va a meter el puerto.

El diseño de este módulo se divide en dos bloques.

- En primer lugar, será necesario definir las estructuras de datos con las que se trabajan. Principalmente, hay que construir una clase para representar el puerto de guía de onda. Pero esto hay que hacerlo considerando cómo se deben incorporar antenas en el módulo MOM. Como se detallará, otras antenas de este módulo utilizan unas estructuras de datos comunes que hay que utilizar para que la integración con el programa sea correcta.
- Una vez diseñadas esas estructuras, hay que definir la interfaz de usuario que permitirá al usuario agregar esas antenas y cumplir con el resto de casos de uso propuestos.

Diseño general de las antenas en el módulo MOM

Tal como se ha indicado en numerosas ocasiones dentro de esta memoria, el módulo MOM se compone de distintos tipos de antenas que se puedan usar en las simulaciones.

Eso significa que es necesario modelar la forma en la que cada tipo de antena interactúa con el programa. El módulo MOM dispone de una serie de interfaces y clases que toda antena que quiera agregarse debe implementar.

En primer lugar, la clase que represente una antena deberá extender a la clase abstracta `AntennaGeometry`. newFASANT 6 utiliza esta clase para abstraer la funcionalidad común que permite tratar a las antenas como geometrías que pueden ser representadas en la interfaz gráfica mediante Java3D.

En concreto, a su vez `AntennaGeometry` es una `Shape3D`, que es instanciada y renderizada cuando se construye para que el usuario tenga feedback visual sobre lo que está agregando al escenario. Esta clase es abstracta y define una serie de métodos que deben ser implementados por las clases concretas que definan un tipo particular de antena. Estos métodos son:

`getGeometryAntenna() : BranchGroup`
: Se espera que este método devuelva un `BranchGroup` de Java3D que permita representar en el escenario la antena. Cada vez que se actualice la geometría, este método será invocado, por lo que el proceso de construcción de este BranchGroup deberá ser *en el momento*, ya que lo más probable es que la antena, cuando se invoque este método, no se encuentre igual que estaba la vez anterior que se llamó.

`getAntennaAppearance() : Appearance`
: Este método deberá devolver la apariencia de Java3D de la antena. La antena implementada tiene libertad para definir su apariencia, sin embargo cuando la antena es seleccionada, se utiliza la apariencia estandar que se usa en newFASANT para representar objetos seleccionados, que es el color amarillo. Para que la clase `AntennaGeometry` pueda restablecer la apariencia original de la antena cuando deja de estar seleccionada, es necesario que este método se la proporcione.

`transformAntenna(transformMatrix : Matrix4d)`
: Este método es invocado cuando la antena debe ser transformada. Normalmente esto ocurrirá cuando la geometría a la que está asociada la antena se transforme para que la antena pueda actualizarse de forma coherente. Por ejemplo, si el usuario mueve, rota o escala un objeto que tenga una antena es necesario que la propia antena, incluyendo sus datos y representación, se actualicen también conforme a la nueva posición y tamaño de dicha antena. La matriz de transformación que se proporciona como parámetro es la misma que la que se usa para tranfsormar a la geometría.

Esta descripción ha dejado caer una norma importante sobre las antenas que se diseñan con el módulo MOM: siempre deben asociarse a un objeto, y no pueden existir solas. Cuando el usuario quiere agregar la antena, debe hacerlo sobre un objeto. La

propia interfaz debería obligar al usuario a seleccionar el objeto al que se le quieren agregar las antenas cuando finalmente se implemente el panel que permite agregar antenas.

Para ello, los objetos de la interfaz, canónicamente considerados como instancias de `Objeto3D`, disponen de un mecanismo para que se le asocien antenas, compuesto de una clase más y de una serie de propiedades y métodos.

Esta clase es la segunda clase que debe ser implementada por la suite de antenas que estamos diseñando en esta memoria y, que debe implementar la interfaz `DatosAntennaObjeto`, que nos ofrece la librería interna del módulo MOM.

`DatosAntennaObjeto` es una interfaz que permite almacenar información sobre las antenas asociadas a un objeto. Por un lado, almacena la lista de antenas que se han agregado a ese objeto. Por el otro, contiene una serie de métodos con información común sobre esas antenas, como por ejemplo su tipo.

Cuando el usuario le agrega una antena a un objeto, esa antena es introducida en la lista de antenas que tenga la instancia de `DatosAntennaObjeto` asociada a ese objeto. Si el objeto no tiene datos de antena todavía, esa estructura es creada para poder agregar la antena.

La clase que implemente esa interfaz tendrá la responsabilidad de agregar los siguientes métodos:

`getTipoAntenna() : String`
: Las antenas en newFASANT se representan un por un código alfanumérico que lo identifique ante el resto de antenas que haya en la suite. Este método debe devolver ese nombre que decidamos ponerle a las antenas.

`contains(o : Object) : boolean`
: Debe determinar si una antena está dentro de esta estructura de datos, pese a que su parámetro sea de tipo `Object` de Java.

`isEmpty() : boolean`
: Determina si esta estructura de datos está vacía, es decir, si no hay antenas agregadas a esta lista de antenas.

`remove(o : Object) : boolean`
: Elimina una antena de la lista de antenas que se hayan incorporado a este objeto. Devuelve un valor lógico que es verdadero si se pudo eliminar satisfactoriamente la antena pedida (por ejemplo, porque el parámetro `o` referencie a una antena que sí que esté en la interfaz de usuario. Si el valor devuelto es falso, será porque el objeto que se ha indicado no estaba en la lista de antenas y por lo tanto no había necesidad de eliminarlo.

`getListAntennaGeometry() : List<AntennaGeometry>`
Este método debe devolver la lista de antenas real de esta estructura de datos. Hay una concordancia entre este método y los métodos `isEmpty`, `contains` y `remove`.

`setActive(tipoAntenna : String)`
Este método es utilizado para notificar a la estructura de datos de antena que se ha cambiado el tipo de antena seleccionado en el proyecto. Dentro de un proyecto MOM puede haber objetos con antenas de distinto tipo (coaxiales, microstrip, guías de onda...), pero la interfaz sólo puede trabajar de forma simultánea con un tipo de antena. Si se selecciona un objeto que tenga antenas coaxial, la interfaz se preparará para trabajar con antenas coaxiales. Si se selecciona después un objeto microstrip, la interfaz se prepara para trabajar con antenas microstrip. Las antenas del objeto no se ven afectadas, sólo si la interfaz debe estar lista o no para trabajar con un tipo concreto de antenas. A este método se le pasa como parámetro el código de la antena que se va a trabajar ahora y se espera que la estructura de datos, a partir del código, determine si debe quedarse activa o no. Generalmente, quedará activa si el código de antena indicado es el de la propia antena. En caso contrario, debería quedar inactiva.

`isActive() : boolean`
Devuelve un valor lógico que indica si esta estructura de datos está activa o no.

Después de examinar las distintas implementaciones de `DatosAntennaObjeto` ya incorporadas sobre el módulo MOM, se ha comprobado que todas las implementaciones actuales, además de ser de tipo `DatosAntennaObjeto`, a la vez heredan de la clase `ArrayList<AntennaGeometry>`, por lo que a la vez son consideradas como listas que representan antenas.

Si bien esto vulnera la buena práctica de *composición sobre herencia*, donde se prefiere aplicar composición en vez de relaciones *ES-UN* propias de la herencia de clases, dado que el resto del módulo MOM espera que la clase concreta *sea* una lista, por lo que se usará este diseño por facilitar la integración a pesar de no ser el más óptimo para esta situación.

Una vez la clase que almacene los datos de antena esté creada, se agregará al objeto. Para esto, `Objeto3D` ya tiene una propiedad, `datosAntenna : DatosAntennaObjeto` mediante la cual podemos asociar al objeto una estructura de datos con propiedades de las antenas. Además, hay varios métodos que se usarán para trabajar con estos datos de antena:

`getDatosAntenna() : DatosAntennaObjeto`
Obtiene esos datos de antena.

`setDatosAntenna(datosAntenna : DatosAntennaObjeto`
Establece esos datos de antena.

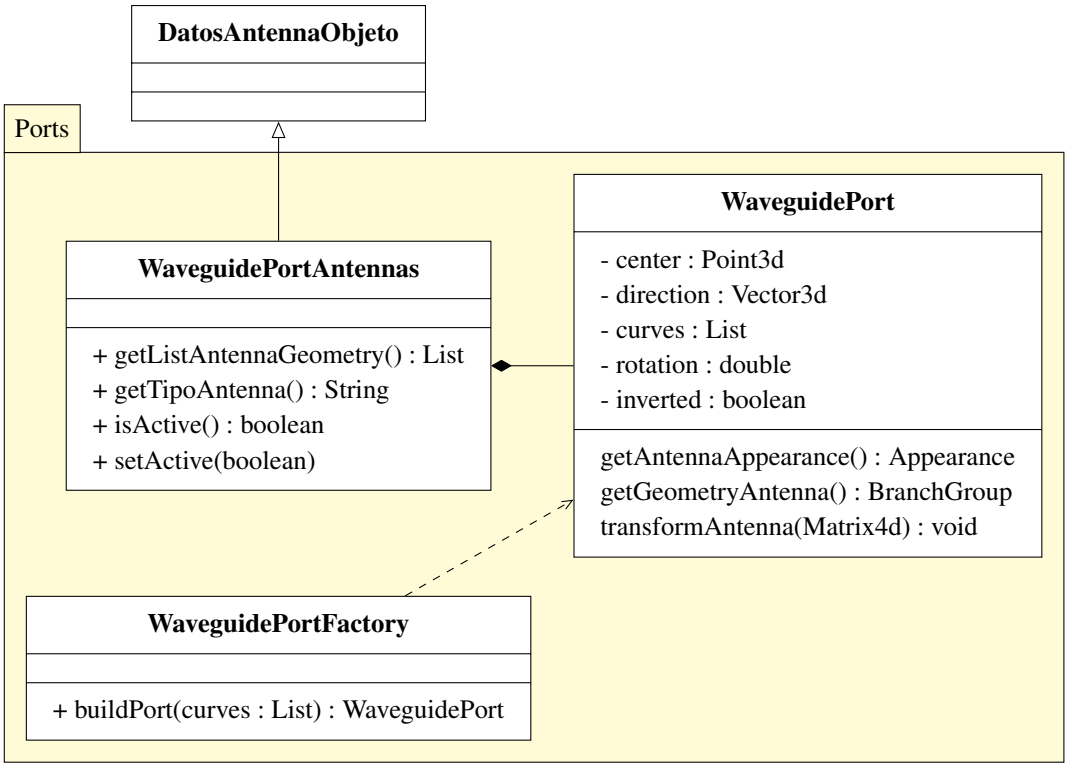


Figura 4.7: Diagrama de clases del paquete de puertos

`isAntenna() : boolean`
: Comprueba si el objeto es una antena. Cuando un objeto tiene asociadas antenas, de cara al resto de la interfaz de usuario es considerado una antena de por sí. Por lo tanto, este método comprobará si existe esa estructura de datos de antenas.

Mención especial para dos métodos que también trabajan con los datos de antena.

`updateJava3DGeometry() : void`
: Cuando este método se ejecuta actualiza la geometría 3D del objeto. Si el objeto es una antena, se debe actualizar la geometría de cada una de las antenas construidas. Tiene que ver con la clase `AntennaGeometry`, cuyos métodos abstractos ya se expusieron anteriormente. Este es el método que invocará al método homónimo que hay en un `AntennaGeometry` que modela la antena.

`transform(transformacion : Matrix4d) : void`
: Cuando se transforma un objeto, si es una antena, deben transformarse también las antenas asocaidas. Esto lo hace invocando a los métodos de transformación de las antenas, y tiene que ver también con el método de la clase `AntennaGeometry` que ha sido mencionado anteriormente también.

Diseño de las estructuras de puerto del módulo de guías de onda

Una vez se ha definido cómo funciona la API de newFASANT para el trabajo con antenas y qué espera el programa de nuestras clases, es momento de establecer el

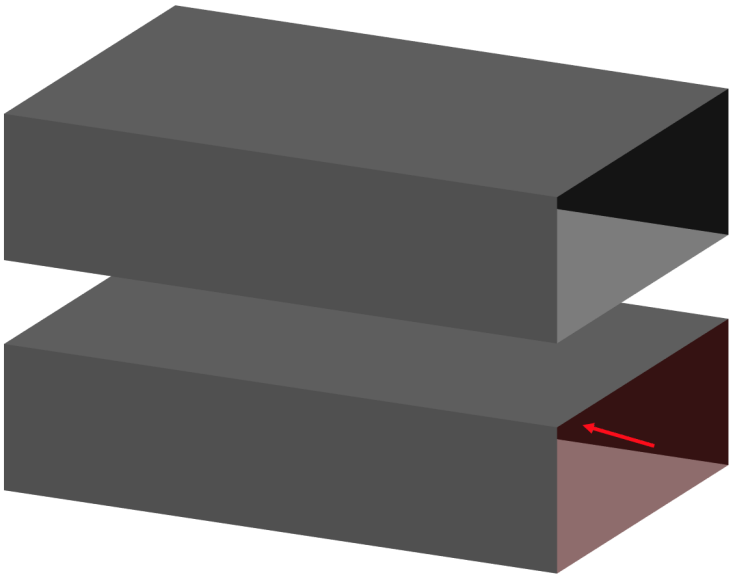


Figura 4.8: Representación de cómo debe visualizarse el puerto agregado a una guía de onda

diseño de esas clases. El modelo completo de este conjunto de clases aparece representado en la figura 4.7. A continuación se desglosa en detalle ese comportamiento.

Comenzaremos definiendo la estructura de datos del puerto, que será lo que de cara al resto del módulo MOM será considerado como *la antena*. La forma en la que el usuario interactúa con el puerto influye en la forma en la que la antena almacena sus datos. Y si el usuario construye puertos mediante la selección de curvas, será necesario que un puerto quede definido por la lista de curvas que ha seleccionado el usuario.

Además, es necesario considerar lo siguiente:

- **Centro geométrico**: a partir de las curvas proporcionadas por el usuario se determina el centro. Esta es una de las propiedades que el kernel requiere conocer para poder ejecutar la simulación ya que determina desde donde se emana la radiación.
- **Dirección de la emanación**: también se determina a partir de las curvas, ya que la emanación siempre es transversal respecto a la sección de la guía de onda. El kernel lo necesitará para determinar la dirección de esa emanación, lo que afectará al acoplamiento contra el resto de antenas del proyecto.
- **Ángulo alfa**: en guías de onda que no sean rectangulares, los puertos pueden tener un valor alfa, mediante el cual el puerto está rotado con respecto al vector director de la emanación.

Pero además, el hecho de que sea una `AntennaGeometry` implica que debemos incorporar en la interfaz de usuario la representación geométrica de esos puertos. El

diagrama representado en la figura 4.8 presenta cómo debería quedar el puerto en la interfaz de usuario.

- Una flecha debería representar el vector dirección para que el usuario sepa la orientación del vector y para que pueda invertir su posición en caso de que no se represente como él espera.
- Un parche debería representar la superficie cubierta por el puerto; la sección que va a ser tratada por el puerto.

Para poder instanciar los puertos se hará uso de un **factory method**. Este patrón de diseño de tipo creacional centraliza la instanciación de nuevos puertos en un método ocupado de hacer esa instanciación. Esto permite controlar cómo se construyen nuevos puertos y a la vez agregar comportamiento propio al proceso de instanciación.

En concreto, tal como se ha detallado antes, el usuario construirá un nuevo puerto por medio de la selección de curvas. Sin embargo, el puerto debe conocer cuál es el centro y cuál es el vector dirección del puerto. En algún moemnto será necesario obtener centro y vector a partir de la lista de curvas y puesto que esto es sólo necesario en el momento de instanciar el puerto, el algoritmo que permite obtener centro y vector puede quedar como responsabilidad para el propio factory method.

4.4.2. Implementación de la alimentación por puertos

Implementación de la estructura de datos de los puertos

La implementación de la estructura de datos `WaveguidePort` comienza de una forma bastante lineal introduciendo las propiedades propuestas en fase diseño y agregando los métodos get y set. Las dos complicaciones de esta clase se centran en dos momentos.

- Durante la instanciación del puerto de guía de onda, porque es necesario determinar el centro y el vector dirección del puerto para introducirlos en el constructor.
- Durante el renderizado de la antena en pantalla. Hay que saber generar la primitiva Java3D que permite representar el puerto en pantalla.

Determinación de centro y vector director Para poder instanciar el puerto de guía de onda es necesario conocer el centro y el vector director en momento de instanciar dicho puerto. Puesto que el usuario proporciona la lista de curvas que ha seleccionado y con los que querrá construir el puerto, nosotros tenemos que inferir centro y vector a partir de la lista de curvas.

Dado que este proceso está ligado a la instanciación, la forma más apropiada es proporcionando un **método Builder**, tal como se indicó en fase de diseño. La clase `WaveguidePortBuilder` dispone de un método estático llamado `buildWaveguide` que, a partir de la lista de curvas, instancia el objeto `WaveguidePort` y lo devuelve.

El proceso para inferir centro y vector dirección pasa por construir un parche Coons temporal. Puesto que este parche es equivalente a la tapa que luego se agrega al puerto durante el mallado y la simulación, determinar el centro y el vector dirección con esta superficie no tiene inconvenientes.

$$N(\overrightarrow{A}, \overrightarrow{B}) = N(\begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix}, \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix}) = \begin{bmatrix} A_y \times B_z - A_z \times B_y \\ A_z \times B_x - A_z \times B_x \\ A_x \times B_y - A_y \times B_x \end{bmatrix} \tag{4.1}$$

- Una vez construida la superficie NURBS, obtener el centro de esa superficie resulta sencillo. Las coordenadas del centro pueden obtenerse a partir de las coordenadas cartesianas equivalentes al punto paramétrico $(0{,}5, 0{,}5)$ de la superficie construida.

- Para obtener el vector normal tendremos que obtener dos vectores ortogonales en la superficie NURBS, y luego calculando el vector normal a partir de la ecuación 4.1. Los vectores se obtienen del siguiente modo:

  - Primero se obtiene un punto de origen común a ambos vectores, mismamente el centro, con coordenadas paramétricas $(0{,}5, 0{,}5)$.
  - Para obtener el destino del primer vector, se buscarán las coordenadas de un punto cartesiano próximo al centro cuya coordenada paramétrica V sea 0,50.
  - Para obtener el destino del segundo vector, se buscarán las coordenadas del punto cartesiano próximo al centro cuya coordenada paramétrica U sea 0,50.

Una vez disponemos de una forma de calcular el centro de la superficie y el vector normal de la superficie, podemos finalmente definir el código del método constructor. Puesto que estos métodos privados definidos anteriormente tienen únicamente relación con este método builder quedarán en la misma clase que `WaveguidePortBuilder`.

Renderizado de la geometría por pantalla Como se indicó, el puerto se debe representar con un parche y una flecha. El parche indica la superficie de sección cubierta, y la flecha representa la dirección de la emanación.

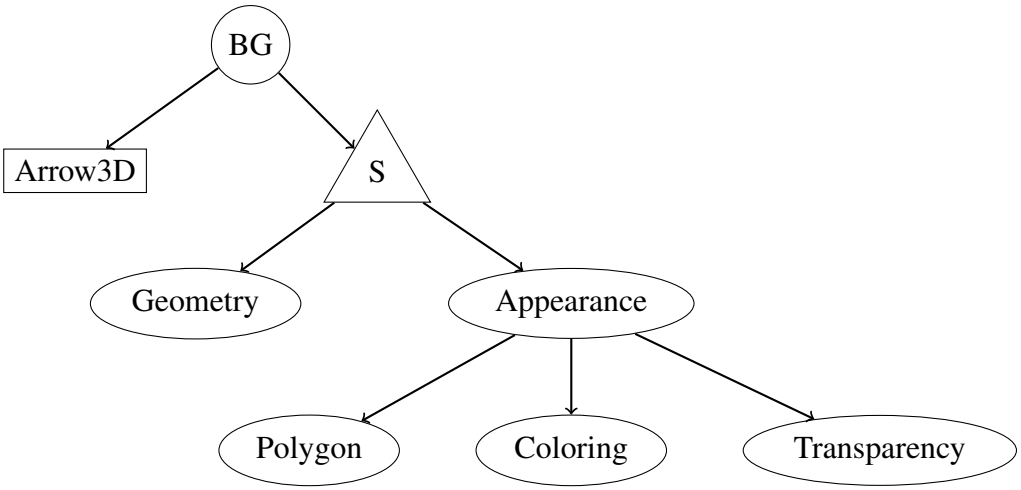


Figura 4.9: Grafo de escena que representa a la antena.

Para implementar esto, se parte de las curvas marcadas por el usuario y que definen el puerto de guía de onda construido y se genera un parche que encierre esas curvas. La operación ya está implementada en la clase `NurbsCreator` y devuelve un objeto de tipo `NurbsSurfaceInterface`.

Sería un error utilizar esta superficie construyendo una `SurfaceShape` porque el objetivo de este parche no es ser representado en el escenario como un `Objeto3D`, sino que debe ser agregado como una primitiva dentro del `BranchGroup` que defina la antena.

En su lugar, lo que se hace es obtener la representación Java3D de la superficie e introducirla. Para esto, después de mirar cómo se hace esto en la clase `NurbsSurfaceShape`, se utiliza un `MeshedSurfaceShape`, que es una clase que utiliza evaluadores para obtener un `GeometryArray`.

Los evaluadores procesan la geometría NURBS contenida por la superficie, convirtiéndola de una representación paramétrica a una representación cartesiana que sea representable en el universo 3D. La `MeshedSurfaceShape` abstrae este proceso y es capaz de devolver el `GeometryArray` de Java3D. Este GeometryArray sí es una primitiva que podemos meter dentro del BranchGroup.

Finalmente, la flecha se puede representar una vez conocemos el vector dirección y el centro. Por medio de la clase de utilidad `Arrow3D` que hay en la librería de geometría de newFASANT se puede obtener una flecha representada de forma interna como un cilindro y un cono. Sin embargo, desde el punto de vista de esta antena es una caja negra.

Finalmente, como geometría de la antena que se devuelva en el método `getGeometryAntenna()` queda un `BranchGroup` compuesto de la flecha y del parche. Una representación del grafo de escena puede verse en la figura 4.9.

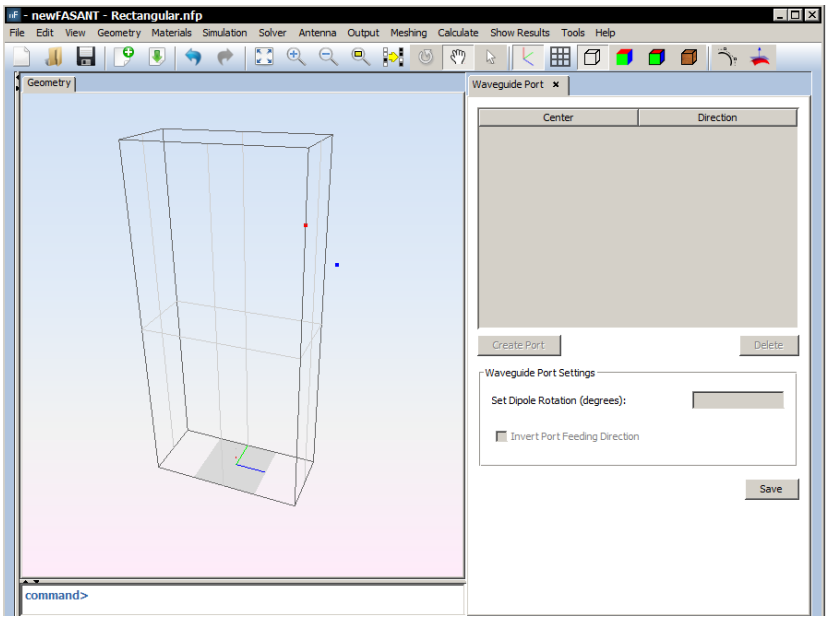


Figura 4.10: Panel para agregar puertos a una guía de onda

Construcción del panel para agregar puertos de guía de onda

Finalmente, el usuario dispone de un panel que utiliza para incorporar a una guía de onda los puertos que necesite. El panel utilizado es visible en la figura 4.10

Cuando el usuario abre este panel seleccionando previamente el objeto que se constituya como guía de onda, podrá seleccionar las curvas que definen el contorno del puerto por el que quiere introducir el puerto de la guía de onda.

Para ello, se identifican las curvas que definen el borde de la geometría y se muestran en pantalla, de modo que cuando el usuario haga clic sobre una de ellas, se coloree para identificarla como seleccionada.

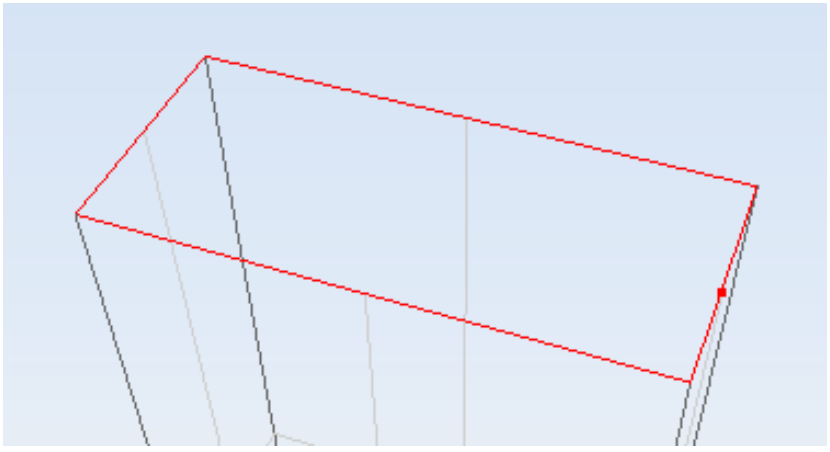


Figura 4.11: Detalle del coloreado de las curvas seleccionadas.

Cuando el usuario selecciona unos bordes cerrados, entonces el botón para crear un puerto se habilita permitiéndole pulsarlo para agregar un puerto a partir de las curvas marcadas, como es visible en la figura 4.12.

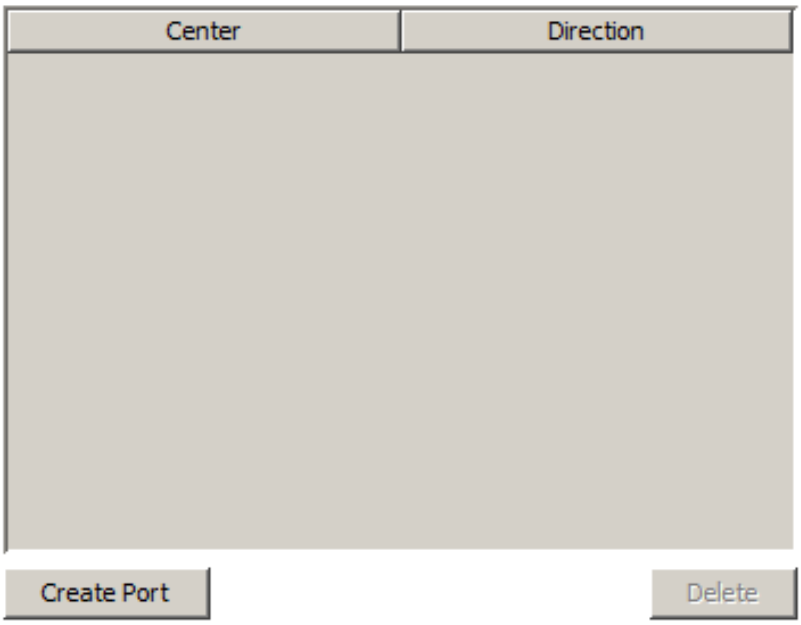


Figura 4.12: Botón para incorporar un puerto a una guía de onda.

Al hacerlo, se introduce el puerto en la geometría, renderizándose tal como fue definido inicialmente mostrando una tapa de color semitransparente que representa el área que será cubierta posteriormente en el circuito y por donde incidirán las ondas, como se ve en la figura 4.13.

4.5. Desarrollo del paquete de ejecución

El proceso de simulación involucra dos etapas. Primero, la generación de la malla por medio del uso del mallador. Después, la ejecución de la simulación. En estas etapas, la interfaz de usuario sirve como apoyo generando los archivos necesarios, preparando

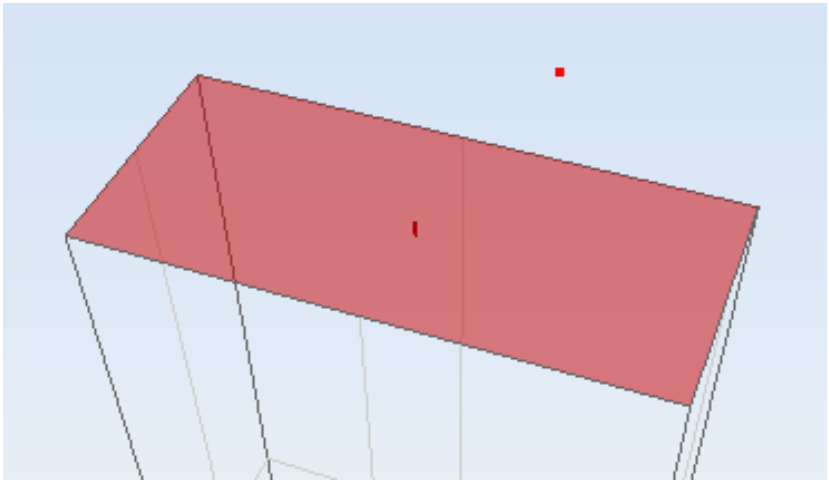


Figura 4.13: Guía de onda con un puerto agregado.

las mallas y realizando las llamadas a los kernels, que son quienes van a hacer los cálculos de verdad.

Es importante considerar llegado este momento que en un proyecto de newFASANT se pueden usar dos configuraciones especiales que pueden afectar a la cantidad de veces que se tendrá que llamar al mallador y al núcleo y por lo tanto al tiempo total empleado en hacer las simulaciones.

- Barrido de frecuencias. Un usuario puede especificar una frecuencia de mallado, pero también puede especificar un rango de frecuencias. Cuando múltiples frecuencias están seleccionadas, se mallará y se simulará el proyecto para cada una de las frecuencias del rango.
- Métodos iterativos (step). Un usuario puede parametrizar el proyecto y establecer que algunas propiedades, como el centro o las dimensiones, estén expresadas en función de un parámetro cuyo valor puede variar. En caso de haber parámetros, se tendrá que realizar un mallado y una simulación para cada iteración.

4.5.1. Trabajo con el mallador

Análisis del mallador

Para poder utilizar los núcleos proporcionados con newFASANT y en concreto el núcleo MONURBS, que es el que se utiliza durante este proyecto, se hace necesario antes realizar un paso previo que es el mallado.

La finalidad del mallado es convertir una representación parametrizada de una geometría basada en NURBS en un modelo discreto optimizado para efectuar simulaciones electromagnéticas. Para ello, el mallador deberá tomar la representación NURBS del proyecto que el usuario quiera simular, y obtener a partir de esa representación una malla que se caracterice por ser **discreta**, es decir, por estar compuesta de una serie de componentes geométricos más simples y reducidos denominados parches.

No obstante, tal y como se acaba de indicar, no sólo es necesario mallar una representación NURBS para obtener otra representación distinta de la geometría. También se hace necesario mallar para poder aplicar una serie de tratamientos que permitan optimizar el proyecto de cara a futuras simulaciones. Debido a que la simulación electromagnética es un proceso con abundante matemática y física, se trata de una operación que consume bastante tiempo de CPU, por lo que se han desarrollado distintas optimizaciones que aplicadas a la malla que se está generando, permitan reducir la cantidad de tiempo de CPU empleada para realizar estas simulaciones.

Durante toda esta sección, y durante el desarrollo, consideraremos al mallador como una caja negra; un programa externo al que se invoca desde la interfaz de usuario proporcionándole unos datos de entrada para obtener una serie de resultados.

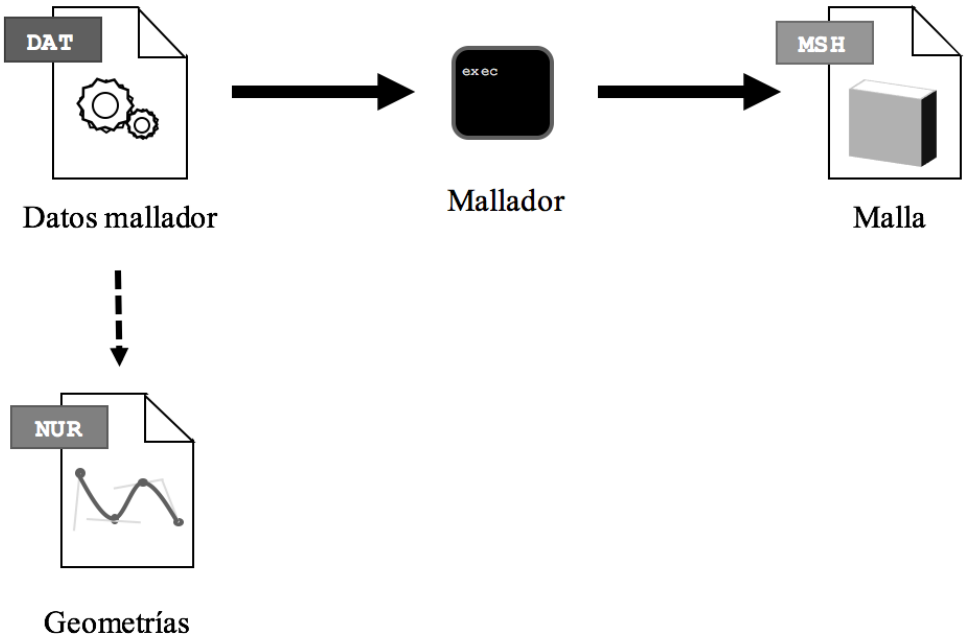


Figura 4.14: Diagrama de entradas y salidas del mallador.

La interfaz principal que tiene este programa para recibir la entrada y para emitir su salida son los archivos. El mallador espera que cierta configuración de entrada se encuentre en algunos archivos que deben ubicarse en el directorio **datos**. Algunos de estos archivos son opcionales y si no están, se generarán automáticamente por el mallador con la configuración mínima como para que no interfieran con las operaciones propiamente llevadas a cabo por dicho mallador o por el kernel que luego realice los cálculos, ya que algunos de los archivos del mallador luego son usados a la vez por el kernel.

Como medida auxiliar, el mallador hace uso de la salida estándar para emitir información sobre el proceso de mallado, como por ejemplo el porcentaje procesado de la geometría, el número de repeticiones en la optimización que se ha hecho o información sobre los errores relativos en los cálculos. En el caso de invocar manualmente el kernel a través de un símbolo de sistema se obtiene la salida impresa por la pantalla. No obstante, la opción más habitual es que el mallador sea un proceso invocado desde la interfaz de usuario. En estos casos la salida estándar suele estar redirigida a la propia interfaz para presentarla de forma personalizada en un log.

Para proporcionar la configuración del mallador se utiliza un archivo de entrada denominado datos_mallador.dat mediante el cual le indicamos los ajustes del proyecto actual. Este archivo se caracteriza por ser un archivo de texto donde cada ajuste está escrito en una línea. Tanto la interfaz de usuario como el propio mallador conocen la estructura del archivo y saben en qué orden tienen que escribir y leer, respectivamente, esos ajustes. La codificación del archivo es ASCII, de modo que los números están escritos como texto ASCII, los parámetros booleanos se representan como letras T/F,

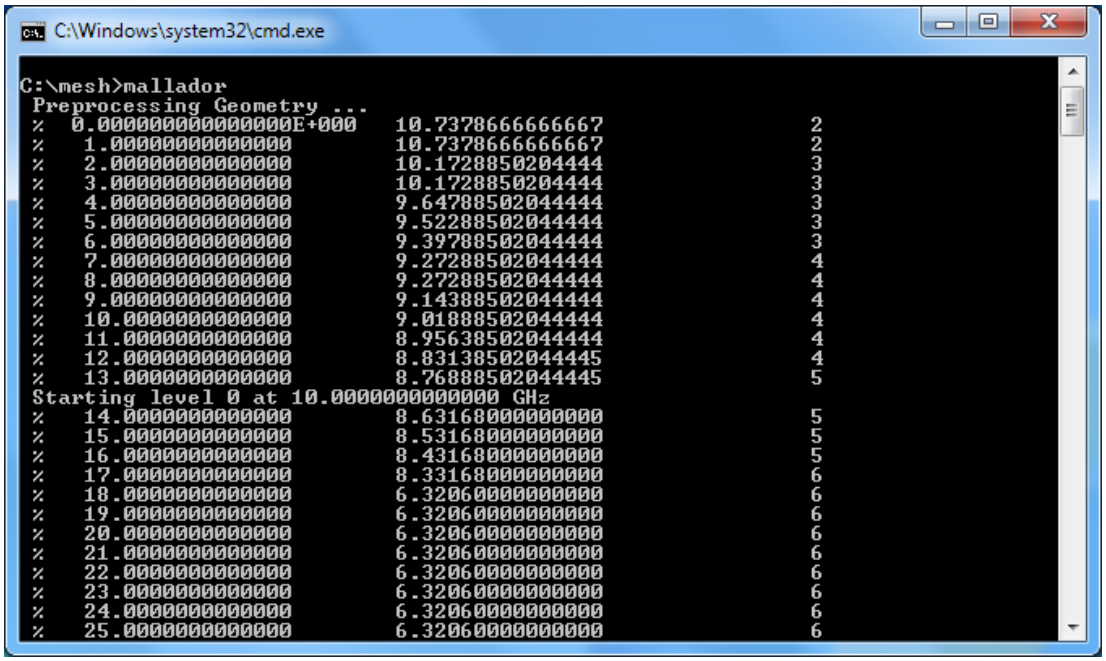


Figura 4.15: Invocación del mallador a través de la línea de comandos.

y las cadenas de caracteres se representan como cadenas de caracteres. En algunas ocasiones un ajuste puede ser compuesto de varias cifras, por ejemplo, `0 0 0 0` para representar cuatro parámetros relacionados que se ha decidido mantener en la misma línea.

Es necesario advertir que el mallador es una herramienta global utilizada por varios módulos por lo que ese archivo de configuración deberá satisfacer a todos esos módulos. En la versión actual del mallador no es posible proporcionar archivos de configuración cuya estructura dependa del tipo de proyecto que se está efectuando (MOM, GTD, IR...), por lo que en este momento ese archivo contiene todos los ajustes de todos los módulos que emplean el mallador. Como resultado, el archivo contiene un montón de líneas de las cuales cada proyecto estará interesado en un subconjunto de ellas.

En el caso de MONURBS, y centrándonos en el tipo de simulaciones que estamos desarrollando en este trabajo, nos interesarán los campos presentados en la tabla 4.1.

Existen otros ajustes que pueden ser modificados por el usuario a través del botón **Advanced Settings**, presentado en el panel **Meshing** que aparece justo antes de que el usuario inicie la operación de mallado, tal como se ve en la figura 4.16. No obstante, no influirán para los procedimientos descritos en esta sección.

El archivo .nur al que se apunta desde la configuración del mallador ha sido previamente generado a través de newFASANT. Contiene una representación de los distintos elementos NURBS de los que se compone el proyecto. El mallador es compatible con NURBS por lo que reconstruyendo las geometrías a partir de sus definiciones es capaz de construir la malla.

Tabla 4.1: Opciones del mallador empleadas en el módulo MOM

Línea	Ajuste	Descripción
1	Ruta	Ubicación en disco del archivo con la representación NURBS de la geometría que se está intentando mallar.
2	Frecuencia	Frecuencia que se va a utilizar para mallar. Influye en la separación de los distintos parches de los que la malla se compone.
3	Divisiones Planas	Número de divisiones por longitud de onda para superficies planas, establecido en las opciones del proyecto.
4	Divisiones Curvas	Número de divisiones por longitud de onda para superficies curvas, establecido en las opciones del proyecto.

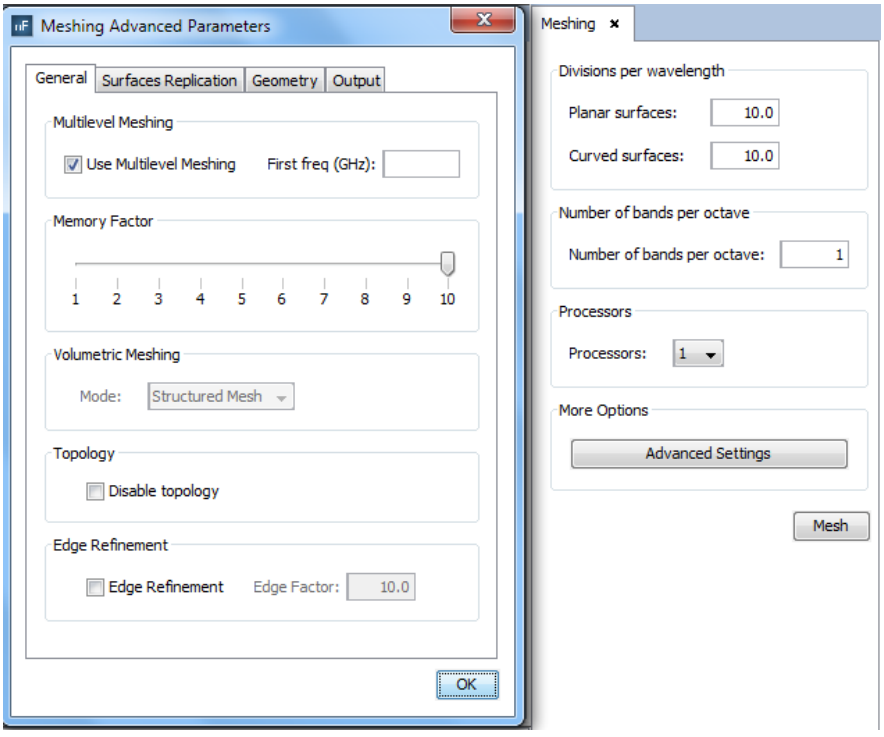


Figura 4.16: Parámetros avanzados del mallador de newFASANT.

Para generar este archivo se hace necesario volcar los distintos parámetros de los que se compone una geometría NURBS. Debido a que es una operación habitual para invocar el mallador ya hay en el código fuente de newFASANT un módulo que guarda las geometrías en un archivo de tipo .nur. El código que haga uso de ese módulo sólo debe proporcionar una lista de las distintas geometrías de tipo ObjectInterface que quiere guardar y la clase se ocupa de generar un archivo estructurado con ellos.

De forma parecida a lo que ocurre con la configuración del mallador, cada uno de los datos a guardar en un archivo de tipo NUR se almacena en una línea codificada en un formato de texto. Tanto el programa que genera el archivo como el programa que los interpreta debe conocer la especificación para poder intercambiar los datos correctamente. El hecho de que sea un formato de texto ofrece al programador una forma sencilla de depurar una geometría a fin de detectar posibles problemas que puedan encontrarse durante el desarrollo.

NUR es un formato de archivos donde se vuelca la información de las superficies NURBS de las que los objetos de la simulación se compone. Está diseñado de modo que permita almacenar la representación de los puntos de control, vectores de nodos y representación de cortes de las distintas superficies de los distintos objetos de la interfaz de usuario.

Preparación de las operaciones de mallado

Aunque el mallador es una caja negra a la que se invoca, es necesario preparar los circuitos que deben ser mallados. Para ello se debe aplicar un algoritmo que permita hacer prolongaciones similar al que se usa en la versión newFASANT 5.

En el módulo MOM de newFASANT 6 ya está implementada la funcionalidad del mallador para trabajar con el resto de funciones del módulo MOM. Lo único que se tendrá que hacer es acoplar todo el código que pertenezca al módulo de guías de onda e integrarlo a la clase principal del mallador.

Para ello se va a diseñar una clase llamada `WaveguideMomWriter` que se ocupe de realizar las geometrías y preparar los circuitos que luego serán mallados. En concreto interesa que esta clase genere lo siguiente:

Circuitos Necesitamos generar los circuitos característicos de cada puerto. Lo que se deberá hacer es localizar todos los puertos del proyecto y para cada uno generar su circuito. El proceso para generar el circuito consiste en producir un archivo de formato NURBS donde se haya volcado la siguiente lista de geometrías:

- El objeto al que pertenece el puerto de la guía de onda principal.

- Una tira que represente el puerto. Esa tira está compuesta de superficies que se marcan como activas, de modo que los kernel traten la tira como una fuente de energía incidente. La tira se debe colocar a una distancia próxima al valor característico de **lambda** para esa guía de onda, respecto a su borde, de modo que la onda tenga que recorrer esa distancia a efectos de formarse correctamente.
- Una prolongación en el borde de la guía de onda sobre la que se sitúa el puerto principal. Se hace para poder encerrar la geometría de la guía de onda, y tiene que hacerse a una distancia mayor que lambda, de modo que no se superpongan las superficies de las tiras con las de la propia prolongación.
- Otras prolongaciones en el resto de bordes por los que se han introducido puertos. También tienen que separarse respecto al borde para que no se superpongan a los bordes de las guías, ya que es ahí donde se harán las mediciones. En este caso el valor de la separación tiene que ser reducido, y se expresa en función de lambda, habitualmente, la vigésima parte de lambda.

El valor de lambda que se necesita para generar los circuitos se corresponde con la longitud de onda de la guía de onda. Es un valor propio de la guía de onda que se calcula en función de la longitud de onda en el espacio libre, que es la que se correspond con la frecuencia que el usuario especifica en las opciones del proyecto. Se calcula a partir de la fórmula:

$$\lambda_g = \frac{\lambda_0}{\sqrt{1-(\frac{\lambda_0}{\lambda_c})}} \tag{4.2}$$

λ_0 es la longitud de onda en el espacio libre y λ_c la longitud de onda correspondiente a la frecuencia de corte de la guía de onda. A partir de este valor, se determinan las longitudes de la prolongación. Las longitudes de las prolongaciones eran modificables por el usuario en newFASANT 5 por medio de unos coeficientes [29]. Aun así, lo normal es usar como coeficiente para la prolongación primaria 1 y para las prolongaciones secundarias $0,02$. El dipolo se sitúa a una distancia de $0,75$ el valor de lambda.

Tapas Además de los circuitos, es necesario construir las tapas asociadas a cada puerto. La tapa es un parche generado a partir de las curvas que definen el contorno que el usuario marcó al seleccionar el puerto. La ventaja de utilizar curvas NURBS en la pantalla de selección de puertos es que la interfaz ya dispone de estas curvas, por lo que debe usar un algoritmo para generar un parche a partir de las curvas.

En la versión 6 de newFASANT se incluye una funcionalidad NURBS para realizar parches a partir de una serie de curvas cerradas, basada en el algoritmo de Coons [30]. Se podrá obtener la tapa aplicando el algoritmo al contorno del puerto. Hecho eso, las tapas se tienen que almacenar en un archivo con formato .nur y proceder a su mallado.

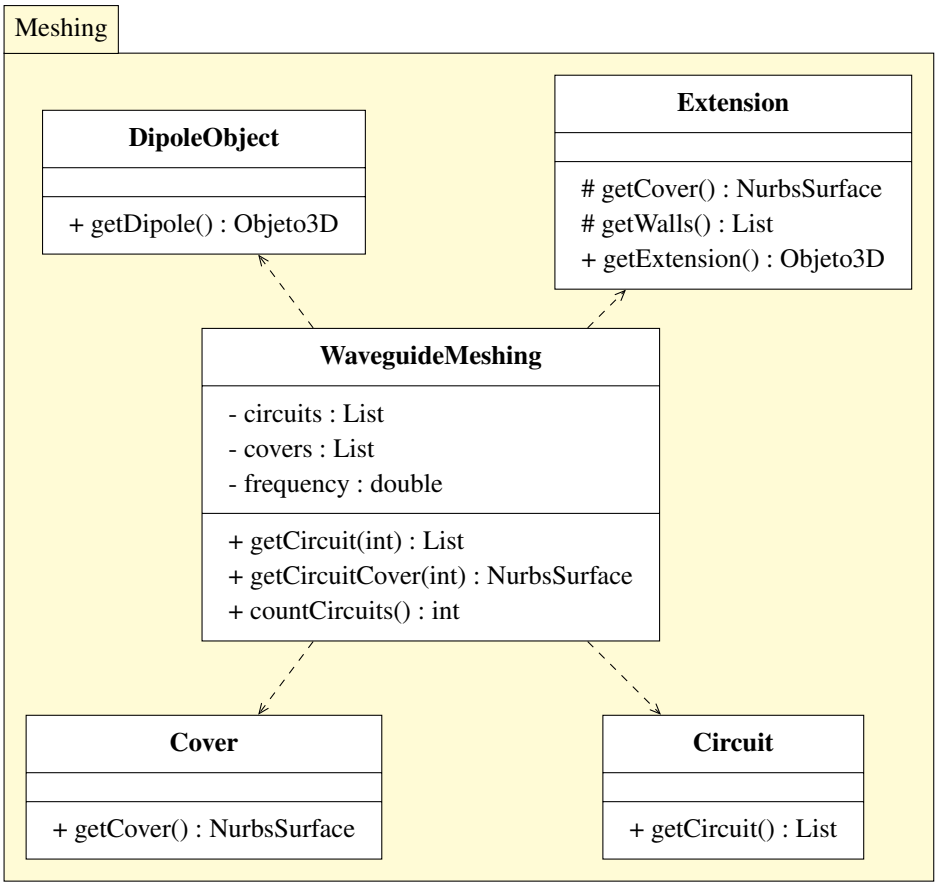


Figura 4.17: Diagrama de clases del paquete de operaciones geométricas de mallado.

Alimentaciones Las alimentaciones son archivos que contienen información sobre las superficies que generan corrientes en cada uno de los circuitos característicos. Para cada circuito hay que localizar las superficies correspondientes a las tiras generadas, y proporcionarles una corriente de 1 V.

Diagrama de clases de la generación de circuitos

Para generar los circuitos se ha elaborado un paquete con clases ocupadas de generar los componentes que se han presentado en la sección anterior. En la figura 4.17 se presenta el diagrama de clases para mallar los componentes.

Sólo la clase `WaveguideMeshing` es pública, y es la que utilizará el mallador para orquestar la construcción de los circuitos y tapas. El resto de clases son privadas en el paquete y son empleadas por la clase principal por conveniencia para estructurar mejor el subsistema.

- La clase `Cover` se ocupa de generar las tapas por medio del algoritmo de Coons. Su método público `getCover()` permite recuperar la tapa construida.
- La clase `Circuit` se ocupa de generar la lista de objetos de un circuito, que se recupera por medio de una llamada a `getCircuit()`.

- La clase `Extension` es la que hace las extensiones. Tiene métodos para obtener paredes de la prolongación y tapa exterior de la prolongación por separado, pero el método más importante es `getExtension()`, que genera un único objeto con las paredes y el borde exterior y que es el que se incorpora también en el circuito.
- `DipoleObject` genera los dipolos en forma de tira que se tienen que incluir en los circuitos.

4.5.2. Trabajo con el kernel MONURBS

Cuando el proyecto se encuentra mallado y listo para ser calculado, se utiliza MONURBS como kernel que recibe la configuración del proyecto y las mallas con las representaciones discretas generadas en la etapa anterior y que genera los resultados.

En concreto, estamos interesados en obtener el análisis de los parámetros de dispersión para poder examinar las matrices S, Y y Z a través de la interfaz de usuario.

MONURBS

MONURBS es un núcleo disponible con la suite newFASANT. Se instala junto a la interfaz de usuario pero no está integrado dentro de la interfaz, sino que se instala en un directorio y es invocado automátiacmente por la interfaz de usuario, de una forma similar a cómo lo hace el mallador.

MONURBS ya fue presentado en el capítulo Estado del arte, previamente en este trabajo de fin de grado. Se trata de una herramienta que se ha conservado en el cambio de newFASANT 5 a newFASANT 6 y que ha recibido mejoras desde entonces para adaptarse a los nuevos requisitos y tipos de proyectos que newFASANT 6 es capaz de producir.

El funcionameinto principal del núcleo MONURBS queda definido por medio de un archivo denominado `datos.dat`. El usuario define aquí las distintas propiedades con las que quiere ajustar el comportamiento del kernel, y en función de las entradas produce unas salidas determinadas. MONURBS trabaja con dos directorios principales:

- `datos`: este directorio es donde la interfaz aporta los datos de entrada, como la configuración con la que se invoca al kernel, o las geometrías que definen la simulación y los datos electromagnéticos que tienen que considerarse.
- `resul`: este directorio es usado para almacenar los archivos generados por MONURBS. Aquí es donde en función del tipo de proyecto que se esté usando se generan unos archivos de salida con un formato que debe ser luego reinterpretado por la interfaz gráfica de cara a presentarlos ante el usuario.

Tabla 4.2: Opciones del kernel MONURBS empleadas en el módulo MOM aplicado al cálculo de S-parámetros

Línea	Ajuste	Descripción
1	Ruta	Ubicación en disco de la malla (.msh) que se va a utilizar para realizar los cálculos.
6	Tipo de cálculo	Qué debe calcular el kernel. En el caso de proyectos de s-parámetros será SPA.
14	Frecuencia	La frecuencia en el espacio libre que se esté usando en ese momento.
19	Divisiones planas	El ajuste de divisiones planas establecido en las opciones de mallado.
20	Divisiones curvas	El ajuste de divisiones curvas establecido en las opciones de mallado.

Matrices de S-parámetros con MONURBS

El procedimiento para calcular las matrices de S-parámetros con MONURBS pasa por establecer en el archivo de configuración de MONURBS el tipo de simulación como `SPA`. El tipo de simulación se introduce en la línea 6 del archivo datos.dat.

El archivo de configuración de MONURBS contiene ajustes que son usados por todos los tipos de proyectos que tienen que usar MONURBS. En consecuencia, algunos cambios no se utilizarán. No obstante, algunos de los campos que se introducen en el archivo de configuración de datos tienen que generarse de acuerdo con los ajustes de la interfaz de usuario.

En la tabla 4.2 se disponen las principales líneas del archivo con la configuración que debe proporcionarse. En la práctica, debido a que MONURBS ya está siendo utilizado por el módulo MOM, prácticamente la totalidad de las líneas ya están escribiendose según los ajustes del proyecto, por lo que lo único que tendrá que establecerse es la ruta de la malla en cada caso en la línea 1 y el tipo de proyecto en la línea 6.

4.5.3. Parámetros de entrada de MONURBS

Cuando MONURBS se ejecute para realizar cálculos de parámetros de dispersión leerá una serie de archivos de entrada. Desde el archivo de datos del kernel se obtendrá la ruta de la malla y se cargará para trabajar con ella. MONURBS necesita ejecutarse una vez por cada circuito que vaya a ser simulado.

Pero además, debe existir archivo denominado `spa.dat` que contiene los datos de entrada usados para hacer el cálculo de los parámetros de dispersión. En este fichero se definen los distintos puertos del proyecto, con sus propiedades.

El formato del archivo es, como muchos de los archivos de newFASANT, de texto. En la primera línea se introducirá el token *SEC* para especificar que el análisis debe hacerse de forma sectorial.

En la segunda línea se especificarán dos números separados por un espacio. Primero, cuántos puertos hay en este proyecto, y segundo, cuál es el puerto principal asociado a ese archivo de datos. Puesto que tiene que proporcionarse un archivo de datos para cada invocación del mallador, en cada iteración se deberá rotar el número de puerto principal de modo que cuando se ejecute una vez MONURBS para cada circuito que hay que procesar, todos los puertos hayan tenido la oportunidad de ser rotados.

Y, a continuación, habrá que indicar las propiedades de cada puerto que haya en el proyecto. Cada puerto requiere una serie de líneas y se tienen que almacenar las líneas de datos de todos los puertos comenzando por el puerto que tenga como identificador 0, y en orden secuencial hasta el último puerto.

Las primeras tres líneas de cada puerto definen el tipo de guía de onda sobre el que se acopla el puerto. En la primera línea se indica si es circular o rectangular mediante los tokens CIR y REC, respectivamente. A continuación se indica el modo dominante, que será `1,0` en el caso de guías rectangulares y `1,1` en el caso de guías circulares. Y en la tercera línea, las dimensiones características de la guía de onda. Si es una guía rectangular se indicarán los valores de A y B, y si es circular, los de R.

Las siguientes dos líneas contienen propiedades del puerto de guía de onda. En la cuarta línea se indica el factor lambda de la guía de onda. A continuación se indica la ubicación al archivo de malla que contiene la tapa que fue creada durante el mallado del proyecto.

Las últimas cuatro líneas de cada puerto definirán su posición y orientación. La sexta línea contiene las componentes X, Y, Z del centro en el que está ubicado el puerto de guía de onda, y es desde donde se harán las emanaciones de energía electromagnética. Las últimas tres líneas se corresponde con la matriz de cosenos directores del puerto.

La existencia de la matriz de cosenos directores se basa en el hecho de que el kernel MONURBS tenga planteadas las ecuaciones para calcular los parámetros de dispersión de un circuito asumiendo que el plano transversal de la guía de onda es el XY; es decir, que la onda avanza en la dirección del eje Z. Las matrices de cosenos directores se utilizan para rotar las geometrías desde una posición arbitraria hasta los ejes de referencia que MONURBS espera.

- El eje Z es el vector director del puerto de guía de onda real construido por el usuario. Como se ha dicho, MONURBS utiliza como referencia una onda que avanza longitudinal al eje Z, de ahí que sea necesario para MONURBS rotar los circuitos de modo que estén alineados.

- El eje Y es el vector de dirección del contorno. Se calcula a partir del vector de dirección del lado corto en guías de onda rectangulares o del radio de una guía de onda circular.

- El eje X se calcula a partir del producto vectorial $Y \times Z$ debido a que los otros dos son más sencillos de obtener y a que lo único importante es que los vectores directores formen una base entre sí, es decir, que sean ortogonales.

Archivos de salida de MONURBS

Para el cálculo de matrices de dispersión, interesa el archivo `spa_dat.out`, que contiene los valores de las matrices de admitancia y las impedancias teóricas para cada par de puertos i, j del proyecto.

Cada vez que se invoque el núcleo MONURBS se generará este archivo, donde se estudia, para el puerto principal que se haya indicado al kernel, la admitancia entre el puerto principal y todos los demás puertos del circuito. Es por ello que habrá que preservar este archivo de modo que contenga todos los valores del proyecto.

Para ello, se leerán los valores de Y y el valor de Z_0 obtenidos con MONURBS y se introducirán en un archivo que agregará todos los parámetros Y y todos los parámetros Z_0. Este archivo tendrá un formato sencillo de procesar (se va a optar por CSV), de modo que en la fase de análisis de resultados se pueda leer el archivo y generar las matrices Z y S a partir de él.

Dentro del archivo `spa_dat.out`, la información de cada puerto ocupará unas 5 líneas. En consecuencia, los archivos `spa_dat.out` tendrán una longitud de $5n$ líneas, donde n es el número de puertos del circuito. De entre cada grupo de 5 líneas, en particular nos interesan la cuarta y la quinta línea.

- La cuarta línea contiene el número complejo que representa el coeficiente Y_{ij} para el puerto principal i y el puerto secundario j..

- La quinta línea contiene el valor de la impedancia teórica Z_0 entre ambos puertos.

Se interpretará cada grupo de 5 líneas y se generará a cambio una línea en formato CSV donde habrá cinco campos separados por comas: número de puerto principal, número de puerto secundario, parte real del coeficiente Y, parte imaginaria del coeficiente Y, y valor de Z_0. Esta línea se escribirá en un archivo llamado *Sparam.out* que será el que almacene los datos de forma persistente como un archivo generado por el proyecto. Como este archivo contiene la combinación de todos los puertos del proyecto, sólo será necesario un único archivo para representar los parámetros en un step y en una frecuencia.

4.6. Desarrrollo del paquete de resultados

En la última etapa de este proyecto, se busca ofrecer al usuario un análisis de las matrices de dispersión, admitancia e impedancia del circuito multipuerto que previamente ha sido mallado y simulado.

Anteriormente, tras la invocación de MONURBS, se habrán obtenido los archivos de datos con los valores de las admitancias e impedancias teóricas para cada par de puertos de los que se compone la simulación que ha sido creada por el usuario.

Ahora, se hace necesario interpretar esos valores y mostrárselos al usuario. Nuestros objetivos para esta sección, por lo tanto, es satisfacer los siguientes requisitos.

- Permitir al usuario examinar la matriz de parámetros S.
- Permitir al usuario examinar la matriz de parámetros Y.
- Permitir al usuario examinar la matriz de parámetros Z.
- Ofrecer representación de los parámetros en su forma binómica (parte real e imaginaria).
- Ofrecer representación de los parámetros en su forma polar (magnitud y fase).
- Ofrecer una valoración de la magnitud de los parámetros expresada en dB.

Todas las matrices deben poder visualizarse de los tres modos. Puesto que todas las matrices se componen de números complejos, la forma de obtener los parámetros en una forma u otra es la misma en todos los casos.

4.6.1. Diseño del visor de resultados

El módulo MOM dispone ya de un visor de resultados de parámetros de dispersión empleado en proyectos de antenas coaxiales. MONURBS genera parámetros de dispersión en proyectos de este tipo, aunque el proceso de cálculo es distinto del utilizado para los proyectos de guías de onda. No obstante, el resultado final también es una matriz de admitancias que debe ser procesada para obtener las matrices S y Z.

No obstante, esos parámetros se representan en un panel que facilita la visualización de los datos permitiendo al usuario seleccionar en todo momento la antena coaxial que se quiere consultar y recibiendo los parámetros solicitados. Un ejemplo de uso de este panel se puede ver en la figura 4.19.

El objetivo de esta sección será adaptar ese panel para que pueda trabajar también con antenas de guía de onda, para lo cual se deberán aplicar ciertas refactorizaciones al código actual de ese panel y a las clases que calculan los datos de las matrices y que alimentan al panel.

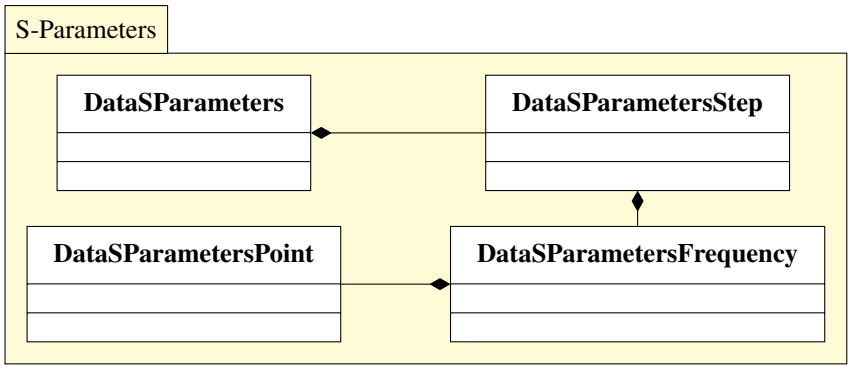


Figura 4.18: Diagrama de clases del antiguo paquete de parámetros S

Estado inicial del análisis de datos

En este momento el análisis de los S-parámetros depende de cuatro clases.

- `DataSParametersPoint`: contiene la representación de los valores de una posición concreta de las matrices.
- `DataSParametersFrequency`: contiene la matriz de todos los valores para una determinada frecuencia de simulación. Estará compuesta de una matriz de instancias de tipo `DataSParametersPoint`.
- `DataSParametersStep`: contiene las matrices de parámetros calculadas a distintas frecuencias para un determinado step en el proyecto.
- `DataSParameters`: esta es la clase principal que se ocupa de leer los archivos de datos, cargar los parámetros-Y y generar las matrices de parámetros-S y parámetros-Y. Se compone de objetos de tipo `DataSParametersStep`, uno para cada step.

Estas clases se representan en la interfaz de usuario en el panel. Los siguientes métodos son utilizados en el panel para obtener los resultados y para presentar otra información de interés en los controles.

- El método `getMatrixY()` de la clase `DataSParameters` se usa para cargar los datos de la matriz Y desde el archivo.
- El método `getMatrixZ()` de la misma clase se usa para obtener la matriz Z.
- El método `getMatrixS()` se usa por causas similares.
- Pero además, cada uno de los objetos de tipo `DataSParametersStep` que hay asociados a los datos es recorrido a fin de obtener todas las frecuencias para las que hay datos.

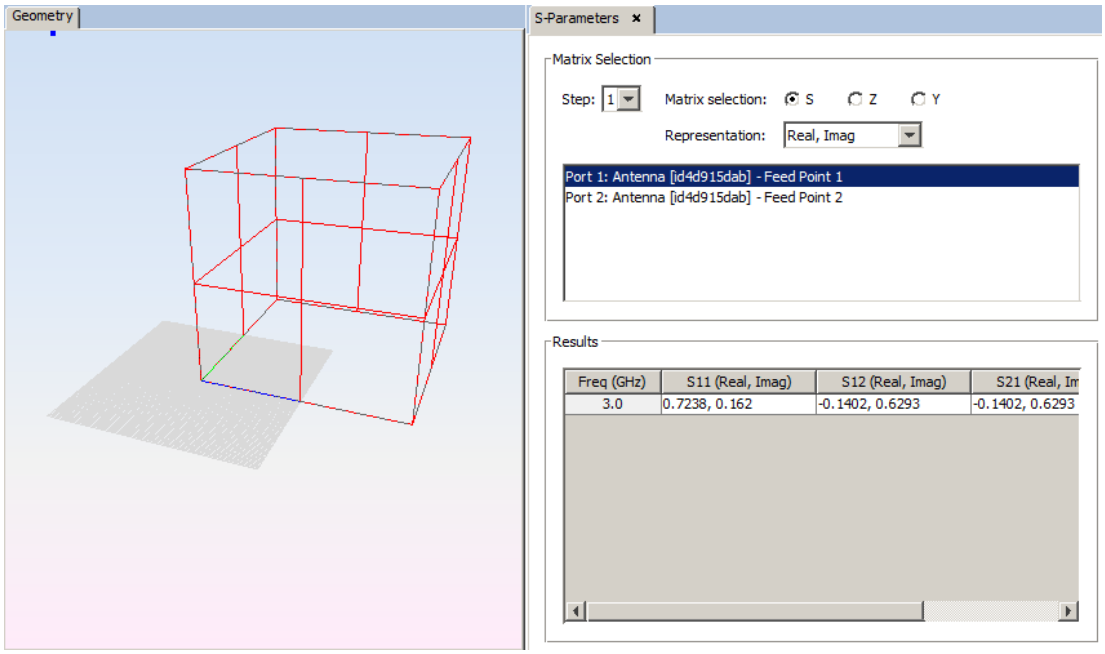


Figura 4.19: Panel de resultados para visualizar matrices de S-parámetros en antenas coaxiales.

Objetivos de la refactorización

El objetivo principal es introducir una capa de indirección entre las clases anteriormente enumeradas y el panel de resultados que no dependa del tipo de antenas del que se obtengan los parámetros de dispersión.

Para ello, se busca introducir una interfaz que permita obtener el listado de frecuencias disponibles para un determinado step efectuado con la interfaz de usuario, y obtener las distintas matrices S, Y, Z en función del step y de la frecuencia seleccionada de todas las disponibles.

Después, se construirá una implementación de esa interfaz que haga esa indirección con respecto a las clases que extraen los S-parámetros para antenas coaxiales, y se usarán esos métodos en el panel en vez de los que ofrece la clase de antenas coaxiales. De este modo, antes de iniciar el diseño propio se verificará que no se introducen fallos de regresión en el código que ya había hecho en el módulo MOM.

Diseño final del sistema

Una vez se aplican esos cambios, el diagrama de clases queda reflejado en la figura 4.20. Las clases que obtienen los S-parámetros en proyectos de antenas coaxiales se han movido a un subpaquete, y es de esperar que la implementación que obtenga los S-parámetros en proyectos de guías de onda desarrollada en esta memoria también utilice un subpaquete.

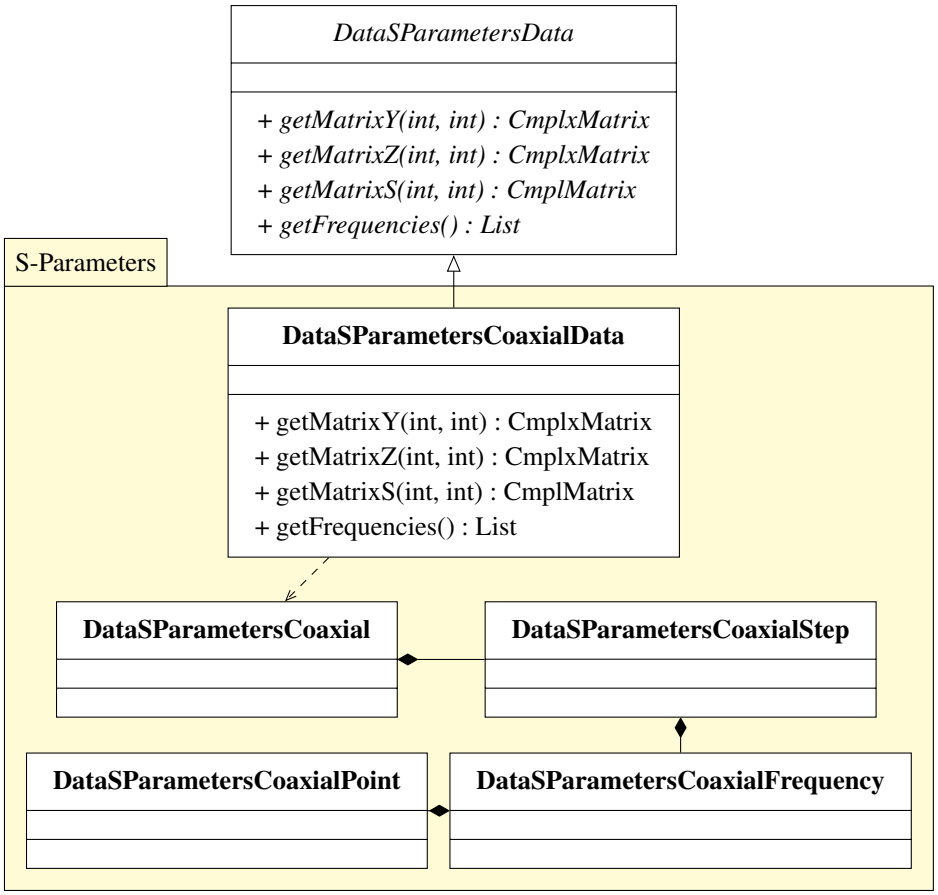


Figura 4.20: Diagrama de clases del antiguo paquete de parámetros S

Diseño particular del análisis de parámetros en proyectos de guías de onda

Una vez estos cambios han sido introducidos, se pueden introducir las clases que permiten extraer parámetros de dispersión en proyectos de guía de onda. La implementación hecha en este proyecto deberá ser compatible con la interfaz que acaba de ser diseñada, para que puedan ser cargados los parámetros en la interfaz de usuario.

El diseño de clases presentado es el siguiente.

- `DataSParametersPointWaveguide` es la clase que mantiene los parámetros de admitancia y la impedancia teórica entre dos puertos. Tal cual, son los valores que se leen del archivo `Sparam.out`, generado anteriormente.
- `DataSParametersFrequencyWaveguide` es la clase que representa una matriz de puntos para una frecuencia concreta. Contiene una matriz bidimensional de instancias de tipo `DataSParametersPointWaveguide`.
- `DataSParametersStepWaveguide` es la clase que representa las matrices de puntos asociadas a un determinado step. Cada una de las matrices se habrá generado a una frecuencia determinada.

- `DataSParametersWaveguide` es la clase que lee los archivos `Sparam.out` del proyecto, y a partir de ellos genera todas las matrices necesarias. Además, dispone de métodos para obtener una matriz Y concreta, Z concreta o S concreta para una determinada frecuencia y step.
- `DataSParametersDataWaveguide` es la implementación de la interfaz que permite obtener las matrices y los valores de las frecuencias. El panel se conectará con esta clase como intermediaria para obtener las matrices.

4.6.2. Implementación de resultados

El principal problema a resolver en la implementación es la carga de las matrices desde los archivos generados en MONURBS. Puesto que esos archivos sólo contienen los valores complejos de la matriz de admitancia hay que generar las otras dos matrices en la interfaz.

Generación de la matriz Z

Obtener la matriz Z conocida la matriz Y es trivial debido a que la matriz de admitancias es inversa respecto a la matriz de impedancias. Por lo tanto, para obtener la matriz Z a partir de esa matriz Y se deberá obtener la matriz inversa.

Generación de la matriz S

Para obtener la matriz S hay que conocer previamente la matriz Z y la matriz de impedancias teóricas, de modo que con esta última podamos identificar la impedancia teórica de cualquier par de puertos. Conocidas esas dos matrices, que deberían indicarse como parámetros, procedemos a calcular la matriz S.

Sea I_N una matriz identidad de orden N donde N es el número de puertos de un proyecto (a la vez que el orden de la matriz Z), e $\sqrt{y}$ la matriz de admitancias características, que también es inversa a la matriz de impedancias características, por lo que $sqrty = sqrtz^{-1}$. Podemos determinar $sqrtz$ como [31]:

$$\sqrt{z} = \begin{pmatrix} \sqrt{z_{01}} & & & \\ & \sqrt{z_{02}} & & \\ & & \ddots & \\ & & & \sqrt{z_{0N}} \end{pmatrix} \tag{4.3}$$

Entonces puede determinarse la matriz S como la siguiente expresión matricial:

$$S = (\sqrt{y}Z\sqrt{y} - I_N)(\sqrt{y}Z\sqrt{y} + I_N)^{-1} \tag{4.4}$$

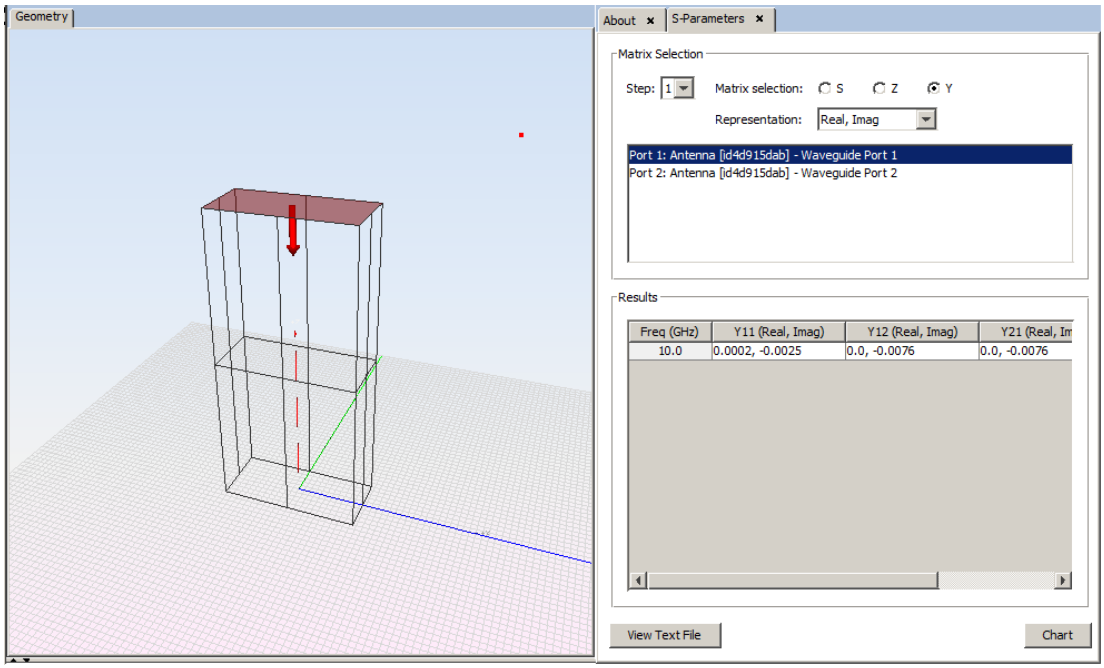


Figura 4.21: Panel de resultados para visualizar matrices de S-parámetros en guías de onda.

Incorporación de los resultados al panel

Una vez se tienen construidas las clases que permiten cargar datos de las matrices de dispersión, sólo hace falta integrar estas clases con el panel para poder visualizar los parámetros de un proyecto de guías de onda. El resultado es visible en la figura 4.21.

5 Conclusiones

5.1. Conclusiones

Una vez se ha terminado de implementar el módulo y agregado las características en el software de simulaciones electromagnéticas, se puede examinar en un plano general el producto desarrollado a fin de obtener una serie de conclusiones.

El objetivo ha sido incorporar una suite para el trabajo con guías de onda y alimentación por puertos sobre el módulo MOM en la interfaz de usuario de newFASANT 6. Se trata de agregar funcionalidad nueva en un software, para lo que es necesario realizar un análisis previo de la situación en la que se encuentra el software y un análisis de cómo está estructurado el sistema de software inicial.

Incorporar las nuevas funcionalidades basadas en guías de onda sobre la interfaz de usuario requiere adquirir una serie de conocimientos sobre guías de onda que conforman el dominio técnico de este módulo. Es importante conocer correctamente los elementos electromagnéticos con los que se está trabajando a fin de obtener una implementación y un diseño limpio donde se sepa verdaderamente lo que se está haciendo. Para el desarrollo de este módulo ha sido necesario estudiar las principales características físicas de las guías de onda y de los parámetros que se estudiarán a partir de las guías de onda.

El módulo desarrollado aprovecha algunas funcionalidades ya disponibles en el software de simulaciones electromagnéticas de newFASANT, como es el uso de los núcleos y del mallador. Tratar estos componentes como cajas negras externas al software ha simplificado bastante el trabajo numérico ya que la interfaz de usuario sólo ha tenido que ocuparse de hacer las llamadas y de comprobar que los archivos generados son válidos.

5.2. Líneas de trabajo futuras

- Implementación de más tipos de guía de onda. En el mercado existen muchos tipos de guía de onda que no han sido presentados en la base teórica debido a que no están tan extendidos. Es el caso de las guías de onda ovaladas, por ejemplo

[9]. Si los procesos de cálculo en este tipo de guías de onda son distintos a los empleados en guías rectangulares o circulares, se podría considerar su incorporación al programa.

- Optimizaciones en el trabajo con guías de onda. Debido a que en el mercado existen guías de onda con tamaños normalizados, se pueden optimizar los comandos para construir las guías de onda ofreciendo la posibilidad de introducir el nombre de una guía de onda estandar en lugar de las dimensiones, de modo que por ejemplo, el usuario pueda indicar que quiere construir una guía WR-90 en vez de indicar las dimensiones exactas de dicha guía de onda.

6 Pliego de condiciones

6.1. Condiciones de obtención del software

El software newFASANT tiene unas condiciones de uso que el usuario final que esté interesado en hacer uso tendrá que cumplir. En particular, newFASANT es un software que debe adquirirse de forma online a través de su página web, `http://www.fasant.com`.

Una vez en el sitio web de newFASANT es posible pedir una versión de evaluación que permita probar el software durante un plazo de 45 días. La versión de evaluación permite probar la mayoría de las características con ciertos límites [1]. Por ejemplo, los proyectos no pueden superar un número de superficies.

Una vez el usuario dispone de su versión de evaluación, se puede poner en contacto con la organización para adquirir la versión completa, la cual se distribuye como una suite común sobre la que se habilitarán los módulos que el usuario haya adquirido. Para poder probar el desarrollo obtenido en esta memoria será necesario adquirir el módulo MOM.

Para poder instalar newFASANT en el ordenador, los requisitos mínimos del ordenador en el que se vaya a ejecutar newFASANT son los descritos en la tabla 6.1.

Tabla 6.1: Requisitos de sistema mínimos de newFASANT [1].

Sistema operativo	Microsoft Windows XP SP2 o superior, o una distribución reciente de GNU/Linux.
Memoria	1 GB de memoria RAM.[I]
Procesador	Intel Pentium Core Duo, AMD Athlon FX o superior.
Disco duro	1 GB de espacio en disco disponible.[II]
GPU	NVIDIA GeForce 8 o superior o ATI Radeon HD 2000
Resolución	1024 x 768 o superior.

[I] La eficiencia del programa será mayor cuanta más memoria disponga la máquina. [II] El espacio real dependerá de los proyectos creados por el usuario.

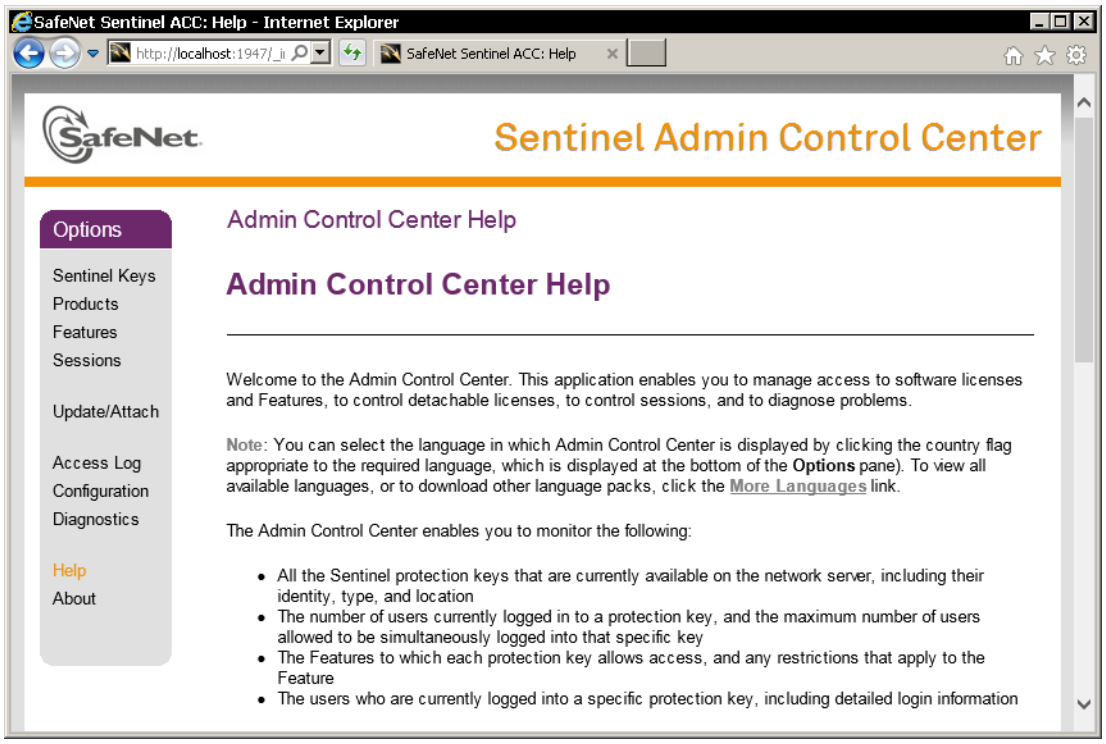


Figura 6.1: Panel de administración del software Sentinel ACC.

Además, puesto que la interfaz de usuario de newFASANT está desarrollada en Java, es necesario contar con el entorno de ejecución de Java, JRE, en su versión 8 o superior, para poder ejecutar correctamente el software de newFASANT.

6.2. Sistema HASP de Sentinel

El software de newFASANT hace uso de la tecnología Sentinel HASP. Sentinel HASP es una tecnología de gestión de derechos digitales que utiliza claves para la protección de software y hardware [32].

El software de newFASANT debe distribuirse junto a un dispositivo USB denominado *dongle* que se debe conectar a un ordenador mediante un puerto USB. Este dispositivo contiene una licencia de uso que depende del tipo de dispositivo que se use.

El ordenador en el que se instale newFASANT debe estar equipado con el software **Sentinel Runtime LDK**, que es el que contiene, tanto el **Sentinel Runtime**, que es el software que reconoce los dongles conectados al ordenador o descubiertos en la red local, como el **Sentinel Admin Control Center**, que es el panel de control que permite ajustar la configuración del Runtime. El Runtime también contiene la interfaz que utiliza el software para comprobar el estado de las licencias.

El software que quiera hacer uso del sistema Sentinel deberá utilizar una librería que se comunica con el Runtime de Sentinel para comprobar el estado de las llaves.

newFASANT utiliza esta librería por lo que para poder utilizar el software es necesario instalar el panel de control de Sentinel como parte del LDK y conectar las llaves al ordenador de forma local o en red.

7 Presupuesto

En esta sección se presenta el presupuesto que determina el coste que supone implementar las funcionalidades descritas. Se presenta el coste tanto de material empleado como de recursos humanos que se necesitan para desarrollar el proyecto.

7.1. Costes de hardware

El hardware tiene que cumplir unas prestaciones que lo hagan apto para poder ejecutar el software de newFASANT y para poder trabajar en su desarrollo. Dado que se trata de un ordenador portátil, sólo se considera el coste del equipo completo.

Tabla 7.1: Costes de hardware para el trabajo desarrollado

Componente	**Precio**
Intel Core i5 2,6 GHz	–,–
Memoria RAM DDR3, 1600 MHz 8 GB	–,–
Disco duro SSD 128 GB	–,–
Pantalla LED 13.3" HiDPI	–,–
Tarjeta gráfica Intel Iris	–,–
Ordenador portátil MacBook Pro Mid-2014	1085,37 EUR
Coste total	**1085,37 EUR**

7.2. Costes de software

Es necesario disponer de un software instalado en el ordenador para poder desarrollar. Muchas de las herramientas empleadas son gratuitas, lo que reduce el coste de software. Para poder instalar el sistema operativo Windows en un ordenador de la familia MacBook Pro es necesario instalar antes el software de Bootcamp, que particiona el disco duro y modifica el gestor de arranque para poder hacer un arranque dual. El software Microsoft Office 2013 se ofrece de forma gratuita gracias a la suscripción de Office 365 en la Universidad de Alcalá.

Tabla 7.2: Costes de software para el trabajo desarrollado

Detalle	**Licencia**	**Precio**
Microsoft Windows 7 Professional	EULA	141.90 EUR
Microsoft Office 2013	EULA	0.00 EUR
NetBeans 8	GPL	0.00 EUR
ArgoUML 0.34	GPL	0.00 EUR
SourceTree	EULA	0.00 EUR
Git	GPL	0.00 EUR
Coste total	**141.90 EUR**	

7.3. Costes de recursos humanos

El coste de desarrollo se asocia al número de horas empleadas para desarrollar las funcionalidades en cada una de las etapas del ciclo del software.

Tabla 7.3: Costes de recursos humanos para el trabajo desarrollado

Tipo	**Horas**	**Precio/Hora**	**Precio**
Análisis de requisitos	60	15.00 EUR	900 EUR
Estudio del software	40	15.00 EUR	600 EUR
Análisis y diseño del sistema a desarrollar	40	20.00 EUR	800 EUR
Implementación y pruebas	90	15.00 EUR	1200 EUR
Documentación	40	15.00 EUR	600 EUR
Coste total			**4100 EUR**

7.4. Coste total

Sumando todos los costes obtenidos se obtiene el coste final del desarrollo del módulo.

Tabla 7.4: Costes totales del proyecto

Item	**Precio**
Costes hardware	1085.37 EUR
Costes software	141.90 EUR
Costes de recursos humanos	4100 EUR
Coste total	**5327,27 EUR**

8 Manual de usuario

El presente manual de usuario servirá como guía de referencia para comprender el funcionamiento del software descrito en esta memoria y del módulo desarrollado durante el mismo.

8.1. Instalación del entorno JRE

La interfaz gráfica de newFASANT está desarrollada en Java, por lo que es necesario disponer del entorno de ejecución Java para poder hacer uso de su máquina virtual y poder ejecutar el software.

El entorno de ejecución Java JRE 8 puede descargarse gratis a través de la página web de Oracle, en su página web [1]. Es importante leer los términos de uso del software JRE 8 si es la primera vez que se instala, y si se está de acuerdo, proceder a pulsar el botón Accept License Agreement para que se pueda iniciar la descarga del entorno JRE 8 en la plataforma para la que se va a utilizar el software.

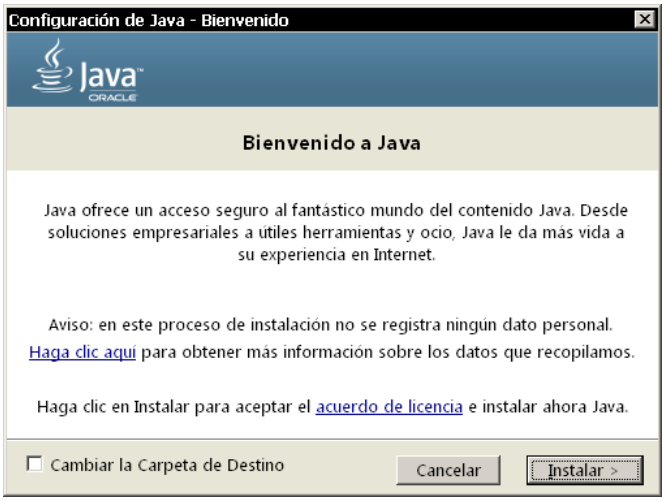
Una vez el instalador se ha descargado de la web de Oracle y se ha ejecutado, el instalador se abrirá y el usuario deberá seguir las instrucciones para instalar el JRE de Oracle. Como se puede ver en la figura 8.1, el instalador únicamente se compone de tres pasos y el usuario sólo necesita interactuar con la primera de las pantallas.

Una vez el software esté instalado en el ordenador, deberá estar dispuesto para ejecutar programas desarrollados con la plataforma Java. Si no fuese posible ejecutar el software en este momento, probablemente sea necesario reiniciar el ordenador para que la configuración del sistema operativo se ajuste correctamente.

8.2. Instalación de newFASANT 6

El software de newFASANT debe obtenerse desde su página web, como ya se describió en el pliego de condiciones. Una vez se ha obtenido un instalador, se puede proceder a su instalación utilizando el asistente de instalación, cuyos pasos se presentan en la figura 8.2.

[1] `http://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html`

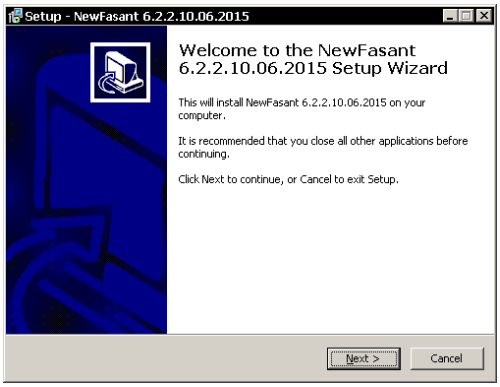


(a) Pantalla de bienvenida del instalador.

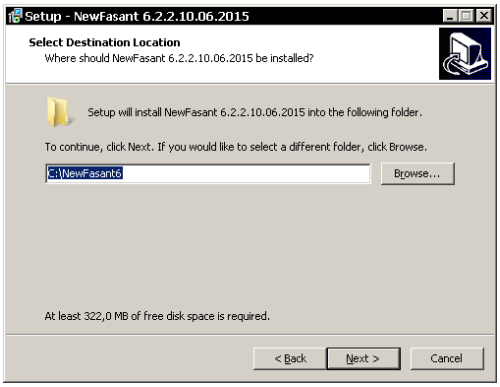
(b) Proceso de instalación del JRE 8.

(c) Ventana de confirmación de la instalación.

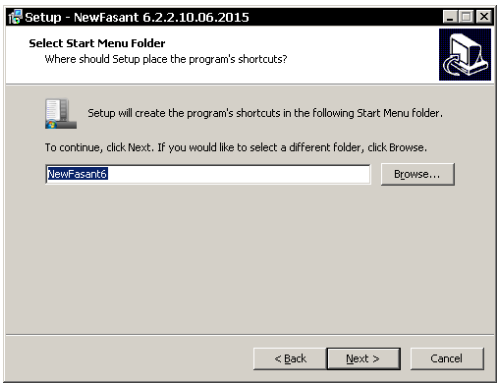
Figura 8.1: Proceso de instalación del JRE 8 de Oracle en Windows.



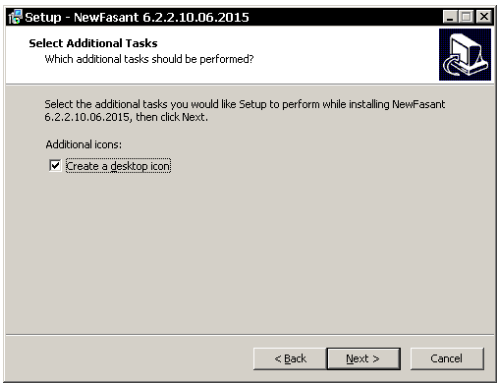
(a) Pantalla de bienvenida del instalador.



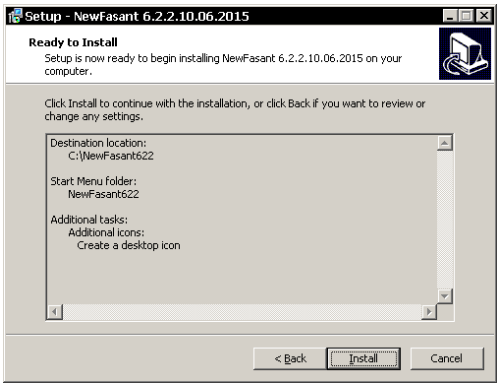
(b) Selección del directorio de instalación.



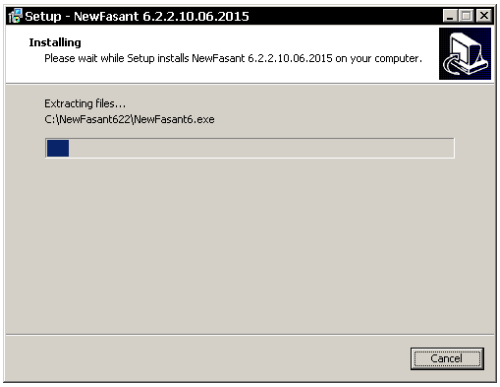
(c) Selección del grupo de programas del menú Inicio.



(d) Confirmación de creación de icono en el escritorio.

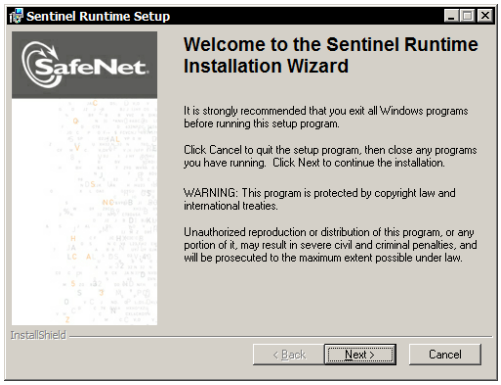


(e) Resumen de los parámetros de instalación.

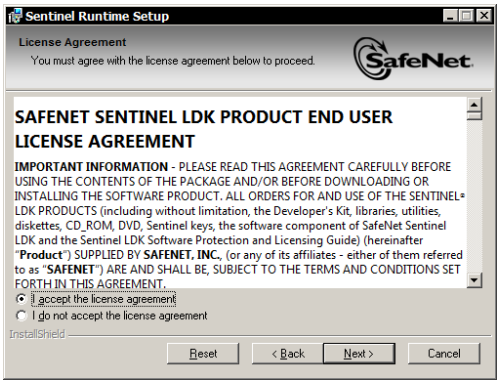


(f) Proceso de instalación de newFASANT 6.

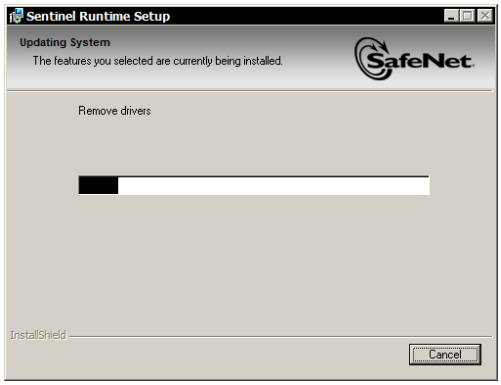
Figura 8.2: Proceso de instalación del software newFASANT 6.



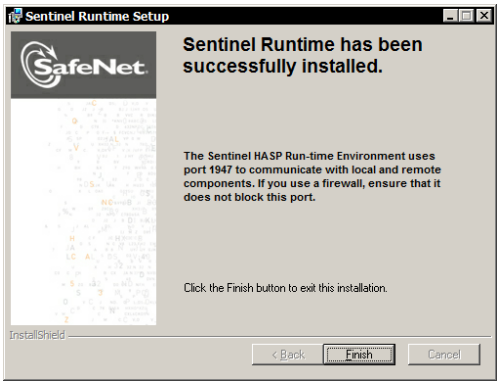
(a) Pantalla de bienvenida del instalador.



(b) Acuerdo de licencia que es necesario aceptar.



(c) Proceso de instalación.



(d) Finalización de la instalación.

Figura 8.3: Proceso de instalación del software newFASANT 6.

8.3. Instalación del Sentinel LDK

Para obtener el `Sentinel LDK` se puede descargar desde el centro de descargas de Sentinel [2]. Desde esa URL se puede descargar el software para la plataforma en la que se esté usando newFASANT. Existe versión para Windows y versión para Linux preparada para las principales distribuciones.

En el caso de Windows se puede obtener un instalador que se ocupe de instalar los componentes del Runtime y los drivers. El asistente de instalación se compone de una serie de pantallas presentadas en la figura 8.3.

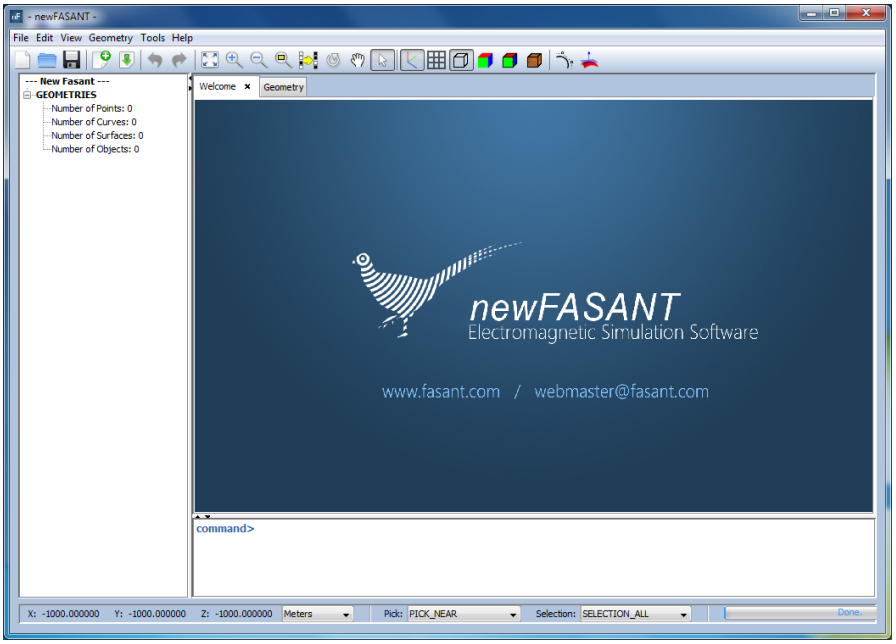
Una vez instalado, puede ser necesario modificar las opciones del Sentinel Runtime. El panel de control de Sentinel es una aplicación web a la que se accede a través de la dirección `http://localhost:1947`. Si el ordenador está equipado con un firewall, puede ser necesario establecer reglas para permitir acceder a este puerto si fuese necesario.

[2] `http://www.sentinelcustomer.safenet-inc.com/sentineldownloads`

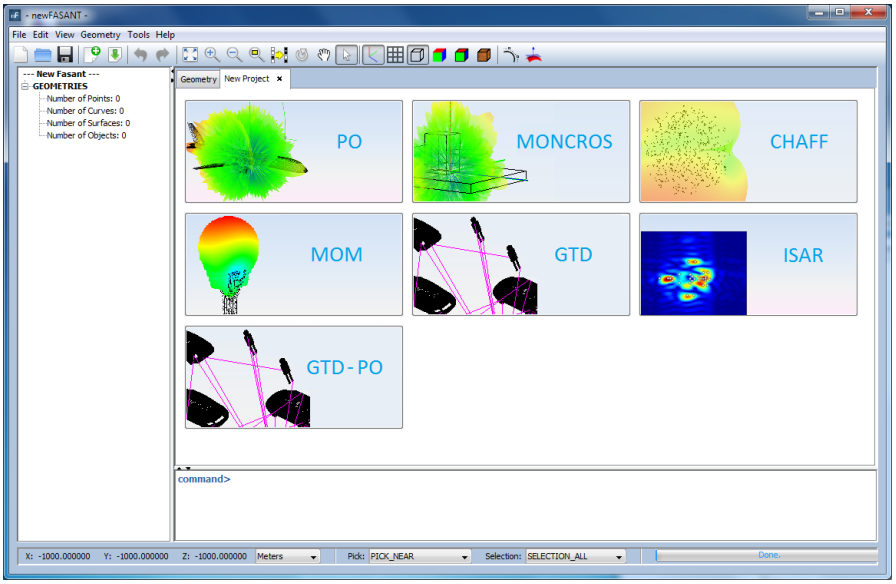
8.4. Empleo de la herramienta de newFASANT

A continuación mostraremos cómo crear un proyecto de newFASANT agregando puertos de guía de onda como se ha implementado en este trabajo.

Una vez abramos newFASANT nos encontraremos con la pantalla principal, con un aspecto parecido al mostrado en la figura 8.4a. Lo primero que haremos es crear un nuevo proyecto de tipo MOM, para lo que iremos a **File — New** y en la ventana de tipo de proyecto construido crearemos un proyecto de tipo MOM.



(a) Pantalla principal de newFASANT 6.



(b) Selección del tipo de proyecto.

Figura 8.4: Primeros pasos empleando la interfaz de newFASANT 6.

Una vez se inicie el proyecto nuevo, se establecen las opciones propias de este proyecto. Se trabajará con una guía de onda de dimensiones 22.86 mm x 10.16 mm, por lo que una de las primeras cosas que hay que ajustar para prevenir posibles problemas, es cambiar la frecuencia del proyecto a un valor que se encuentre en el rango de frecuencias de esta guía de onda para su modo dominante.

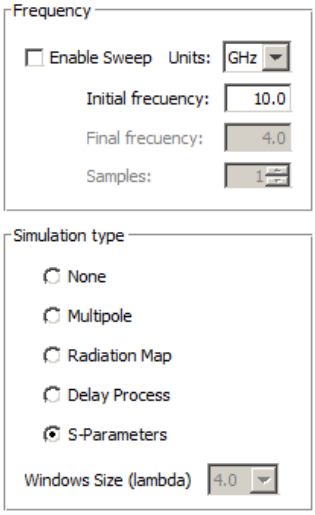


Figura 8.5: Ajuste de los parámetros de simulación en el proyecto.

Esto se ajusta desde el menú **Simulation — Parameters**. En la sección **Frequency** se establece un valor para el que la guía propague en ese modo de operación, por ejemplo, 10 GHz. Además, hay que establecer el tipo de simulación como **S-parámetros**. Hecho eso, se pulsa **Save** para guardar los cambios.

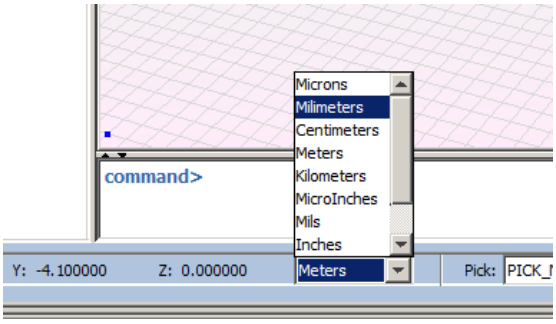


Figura 8.6: Ajuste de las unidades en el proyecto.

Además, se establecen las unidades de este proyecto a milímetros para poder expresar de una forma más cómoda las medidas de este proyecto. En la barra de estado se encuentra el selector de unidades, en el que hay que establecer el valor en milímetros.

Creación de una geometría

Ajustado el proyecto, se procede a crear una guía de onda de ejemplo. En la consola las guías de onda rectangulares se crean con el comando `rectangularWave-`

`guide`. Si no indicamos ningún parámetro, se ejecuta de forma interactiva preguntando los valores del centro, ancho y largo. El usuario puede proporcionar los valores manualmente o aportarlos haciendo clic en la interfaz de usuario para marcar el centro y las dimensiones.

En este ejemplo crearemos una guía de onda de dimensiones 22.86 x 10.16 m. La interacción con la consola es la siguiente:

```
command> rectangularWaveguide
Center of the base [x y z]: 0 0 0
Section size [a b]: 22.86 10.16
Length [double]: 50
```

Con esto se obtiene una guía de onda WR-90 en la interfaz de usuario.

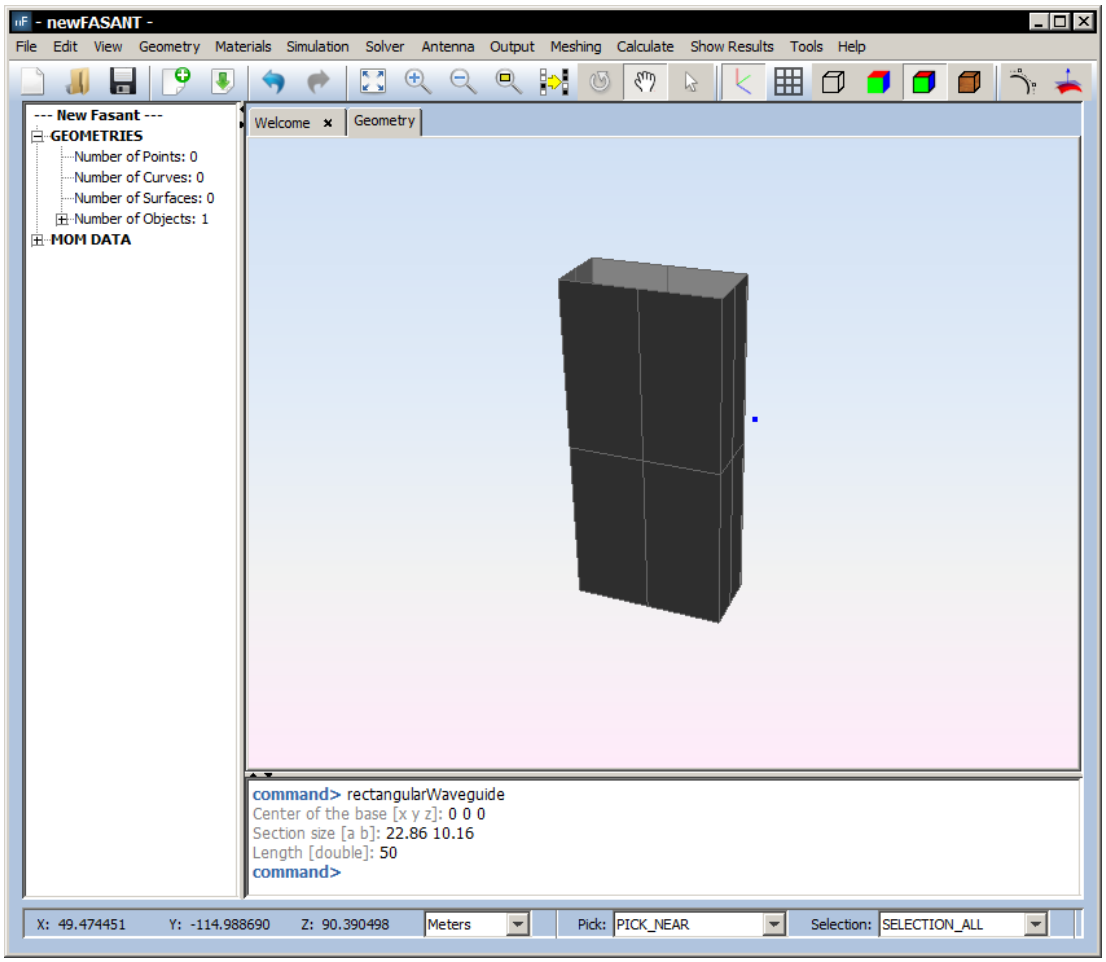


Figura 8.7: Guía de onda construida con el comando `rectangularWaveguide` en la interfaz.

Acoplamiento de los puertos

Ahora se acoplarán a esta guía de onda dos puertos, en los dos extremos de la guía. Para hacer esta operación, se selecciona primero la guía de onda haciendo clic sobre

ella de modo que se ilumine de amarillo, y se selecciona la entrada de menú **Antenna — Waveguides — Waveguide Port**. Con esto se despliega el panel para modificar los puertos.

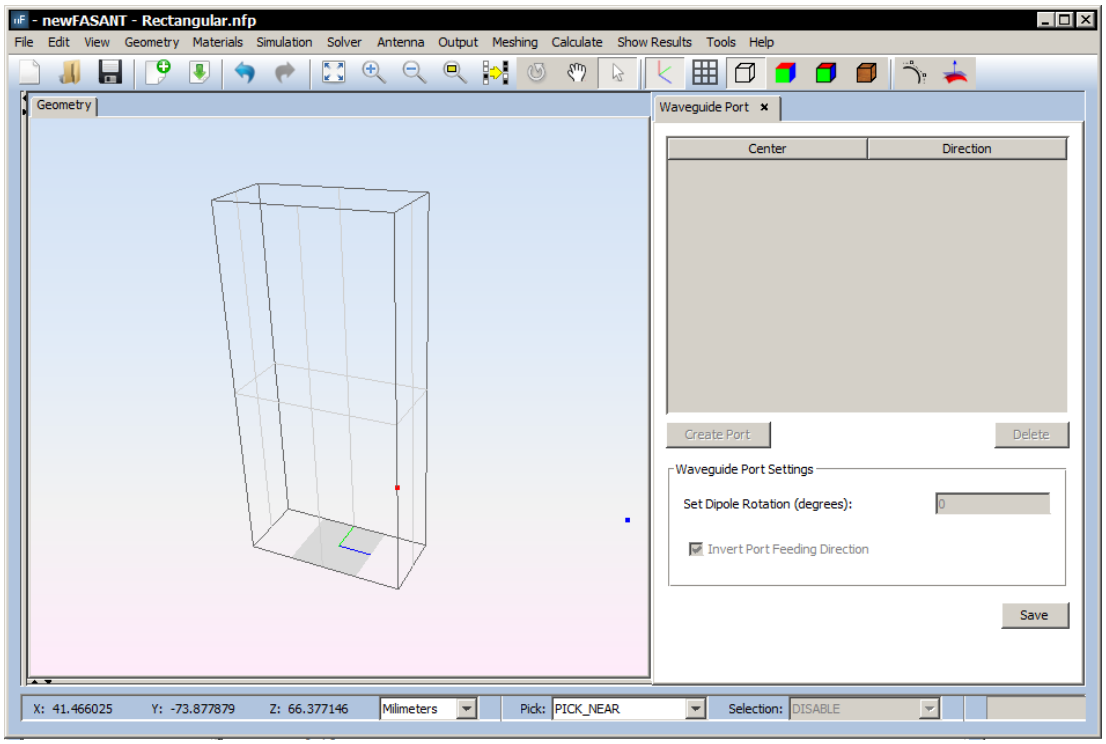


Figura 8.8: Panel para agregar y quitar puertos a guías de onda.

Para agregar un puerto sobre el borde superior, se hace clic con el botón izquierdo del ratón sobre las cuatro aristas de la guía de onda que forman el borde de la parte superior. Cuando se hace clic en una de estas aristas se ilumina de rojo. Manteniendo presionada la tecla Control para que deje seleccionar múltiples aristas, se hace clic sobre cada una de las cuatro hasta tenerlas todas encendidas de color rojo.

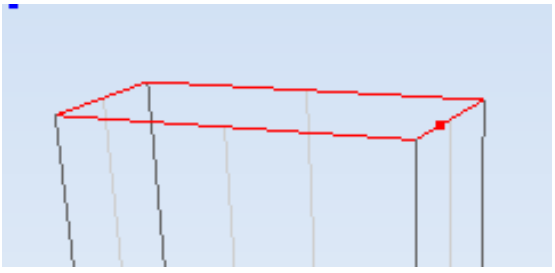


Figura 8.9: Detalle de los bordes iluminándose de rojo al seleccionarlos.

Cuando se seleccionan curvas que delimitan un borde de una guía de onda, el botón **Create Port** se activa de modo que si se pulsa dicho botón, se construye un puerto en esa posición.

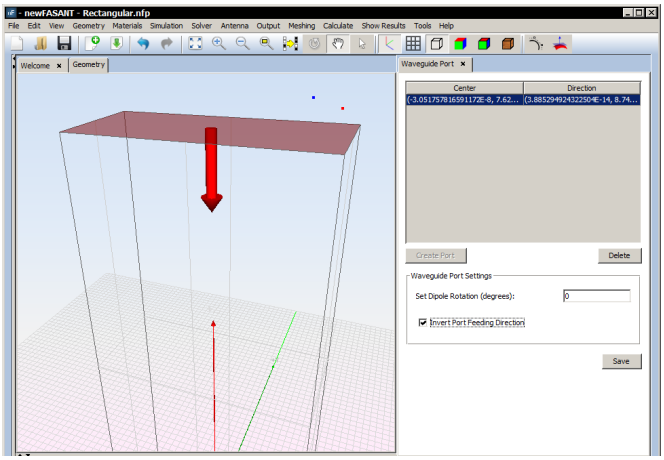


Figura 8.10: Puerto agregado al proyecto.

El puerto se representa coloreando el borde sobre el que se ha definido y representando con una flecha la dirección de la onda que incidirá sobre la guía de onda. Esa dirección debe ser longitudinal a la guía de onda, y debe estar orientada hacia dentro. Si aparece orientada hacia fuera será necesario invertirlo pulsando el check box que hay en el panel para invertir su dirección.

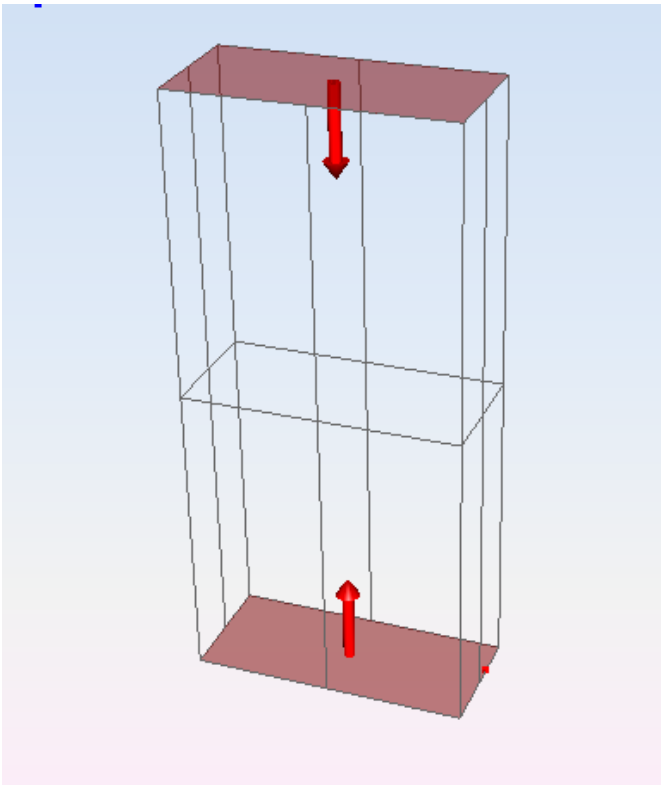


Figura 8.11: Guía de onda a la que se le incorporan dos puertos.

Repetir la operación con el borde inferior para construir un segundo puerto que entre hacia dentro, y después pulsar el botón **Save** para guardar los cambios.

Mallar el proyecto

Para mallar el proyecto hay que ir al menú **Meshing — Create Mesh**. En el panel que se despliega, la opción más importante es la del número de procesadores que se usarán. Es importante que el valor seleccionado coincida con el número de procesadores de la máquina en la que se está ejecutando el software para que se aproveche toda la capacidad de cálculo posible.

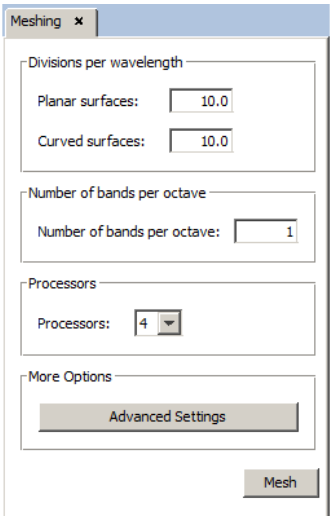


Figura 8.12: Ventana de parámetros de mallado.

Pulsar el botón **Mesh** para iniciar el proceso de mallado, el cual demorará un tiempo en función de la capacidad de cálculo del ordenador. Por el panel de estado se motrará un registro de la información que el mallador emite. Una vez el mallador ha terminado de ejecutarse, se puede cerrar esa pestaña. Los resultados se almacenan en el proyecto por lo que salvo que se cambien los parámetros de la simulación o las geometrías, no será necesario volver a mallar.

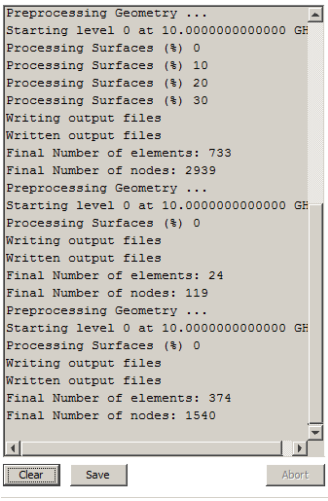


Figura 8.13: Log obtenido por el mallador.

Calcular resultados

Una vez las mallas están construidas es momento de efectuar los cálculos para poder analizar los resultados. Para realizar los cálculos hay que localizar en el menú la opción **Calculate — Execute**. De forma parecida al proceso de mallado, se debe especificar un número de procesadores que coincida con el número de procesadores que tiene la máquina. Si bien no es un hecho crítico usar menos procesos, hará que los cálculos tarden más en obtenerse.

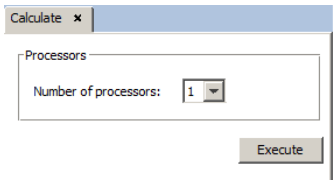


Figura 8.14: Ventana de parámetros de cálculo.

Cuando se pulsa el botón Execute, comienza la obtención de los resultados. De forma parecida a cómo lo hacía el mallador, en este caso se puede visualizar por el log el estado de los cálculos emitido por MONURBS, con datos como el error relativo o el número de iteraciones procesadas.

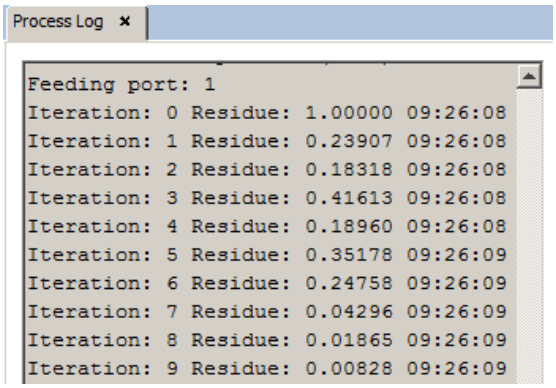


Figura 8.15: Log de salida con los cálculos efectuados.

Evaluación de resultados

Con el menú **Show Results** se pueden examinar los resultados. En el caso de un proyecto de guías de onda habrá quedado activa la opción **S-Parameters**. Haciendo clic sobre ella, podremos examinar los parámetros de dispersión para la guía de onda que se ha mallado y calculado.

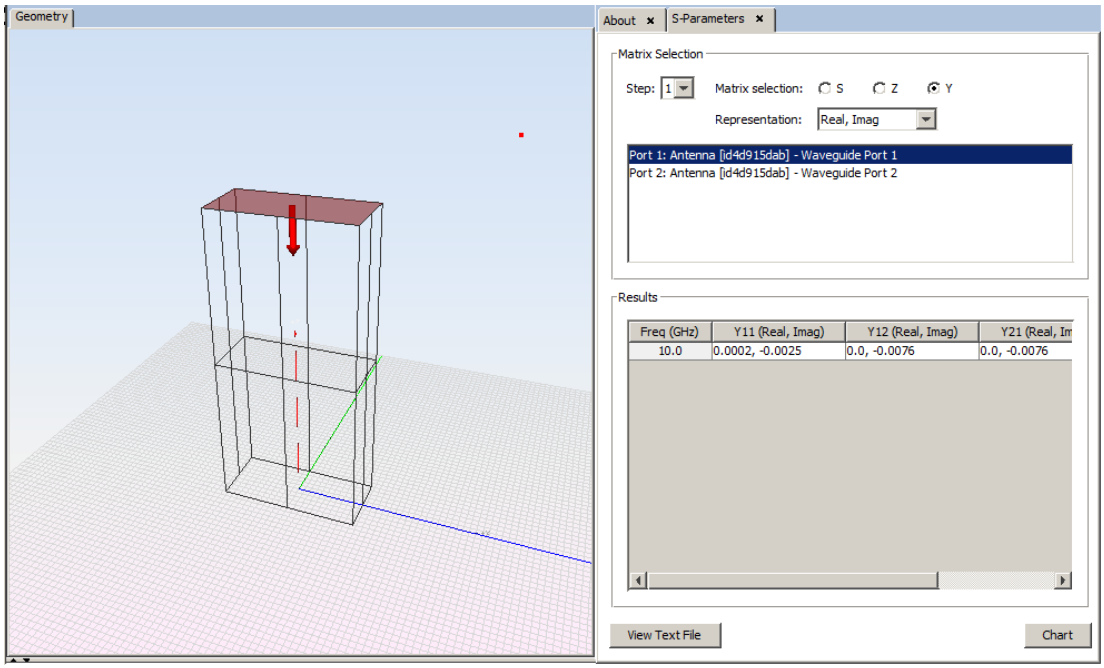


Figura 8.16: Análisis de los resultados obtenidos tras calcular los S-parámetros en la guía de onda.

Bibliografía

[1] newFASANT S.L., "Descargar versión demo de evaluación, v5.7.8."

[2] R. C. Johnson and H. Jasik, *Antenna engineering handbook*. McGraw-Hill Book Company, 1984.

[3] S. J. Orfanidis, *Electromagnetic Waves and Antennas*. Rutgers University, 2002.

[4] W. Tomasi, *Sistemas de comunicaciones electrónicas*. México: Pearson Educación, 2003.

[5] Microwaves101.com, "Waveguide loss." Consultado desde `http://www.microwaves101.com/encyclopedias/waveguide-loss`.

[6] L. D. Simmons and F. L. Ace, *Electronics Technician, Volume 7 - Antennas and Wave Propagation*. Pensacola, Florida: Naval Education and training Professional Development and Technology Center, 1995.

[7] C. A. Ballanis, "Circular waveguides," in *Encyclopedia of RF and Microwave Engineering* (K. Chang, ed.), Tempe, Arizona: Wiiley Online Library, 2005.

[8] D. M. Pozar, *Microwave Engineering*. John Wiley and Sons, 2009.

[9] N. Marcuvitz, *Waveguide Handbook*. Institution of Electrical Engineers, 1986.

[10] D. B. Davidson, *Computational electromagnetics for RF and microwave engineering*. Cambridge University Press, 2005.

[11] Wikipedia, "Electromagnetic field solver — wikipedia, the free encyclopedia," 2014.

[12] CST Computer Simulation Technology AG, "CST MICROWAVE STUDIO." Consultado desde `https://www.cst.com/Products/CSTMWS`.

[13] ANSYS, "ANSYS HFSS." Consultado desde `http://www.ansys.com/es_es/Productos/Flagship+Technology/ANSYS+HFSS`.

[14] Altair Engineering, Inc., "Overview of FEKO." Consultado desde `https://www.feko.info/product-detail/overview-of-feko`.

[15] NewFasant S.L., "CIRCUIT ANALYSIS." Consultado desde `http://www.fasant.com/en/products/7-circuit-analysis`.

[16] I. González, A. Tayebi, J. Gómez, and F. Cátedra, "Monurbs, a parallelized moment method code that combines fmlmp, cbf and mpi," in *EuCAP 2009*, 2009.

[17] A. A. K. Mohsen and A. K. Abdelmageed, "Magnetic field integral equation for electromagnetic scattering by conducting bodies of revolution in layered media," *Progress in Electromagnetics Research*, no. 24, pp. 19–37, 1999.

[18] P. Ylä-Oijala, "Numerical analysis of combined field integral equation formulations for electormagnetic scattering by dielectric and composite objects," *Progress in Electromagnetics Research*, vol. 3, pp. 19–43, 2008.

[19] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, 2015.

[20] ORACLE, "The History of Java Technology."

[21] D. Flanagan, *Java in a nutshell*. Beijing: O'Reilly, 5th ed ed., 2005.

[22] O. Corporation, "The Java® Language Specification."

[23] H. Stahl, "Moving to OpenJDK as the official Java SE 7 Reference Implementation."

[24] Oracle Corporation, "Java SE desktop technologies: Java 3D API." Consultado desde `http://www.oracle.com/technetwork/articles/javase/index-jsp-138252.html`.

[25] J. J. Pratdepadua, *Programación en 3D con Java 3D*. Editorial Ra-Ma, 2003.

[26] H. A. Sowizral and D. R. Nadeau, "An introduction to programming AR and VR applications in Java3D." Consultado desde `http://viz.aset.psu.edu/jack/java3d/java3d.htm`, 1999.

[27] A. Spillner, T. Linz, and H. Schaefer, *Software testing foundations: a study guide for the certified tester exam : foundation level, ISTQB compliant*. Rocky Nook, 4th edition ed., 2014.

[28] MAP, *MÉTRICA Versión 3*. Ministerio de Administraciones Públicas, 2001.

[29] NewFasant, S.L., *newFASANT 5: Waveguides User Guide*. NewFasant, S.L., 2014.

[30] L. Piegl and W. Tiller, *The NURBS book*. Springer Science & Business Media, 2012.

[31] P. Russer, *Electromagnetics, microwave circuit and antenna design for communications engineering*. Artech House, 2003.

[32] S. Inc., “Sentinel HASP HL: Dongles USB Sentinel HASP.”